Chapter 2 - Outline

2.1 Context-Free Grammars

- Formal definition of a context-free grammar
- Examples of context-free grammars
- Designing context-free grammars
- Ambiguity
- Chomsky normal form

2.2 Pushdown Automata

- Formal definition of a pushdown automaton
- Examples of pushdown automata
- Equivalence with context-free grammars

2.3 Non-Context-Free Languages

- The pumping lemma for context-free languages

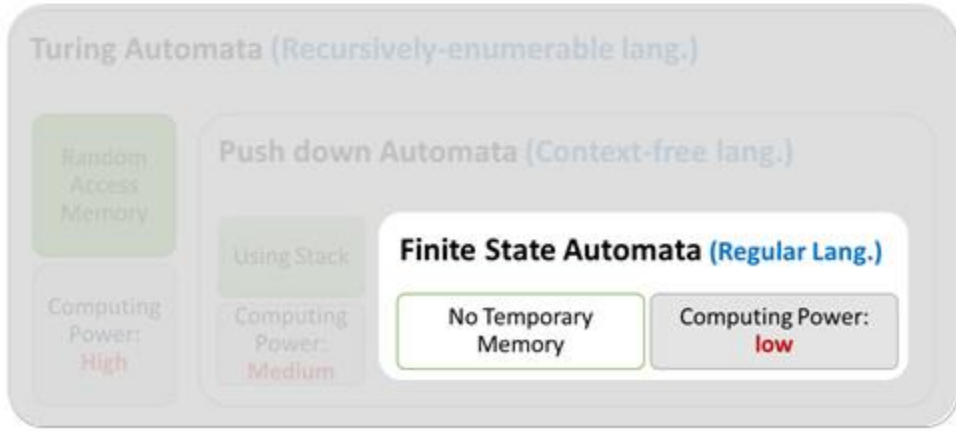
Chapter 2

Introduction

In chapter 1 :

- We learned that
  - we can describe many **languages** (regular languages) using two equivalent methods:
    - finite automata
    - regular expressions

- We found out
  - There are some **languages** that cannot be described using the methods above
    - Non-regular languages
    - Example: $\{0^n1^n \mid n \geq 0\}$



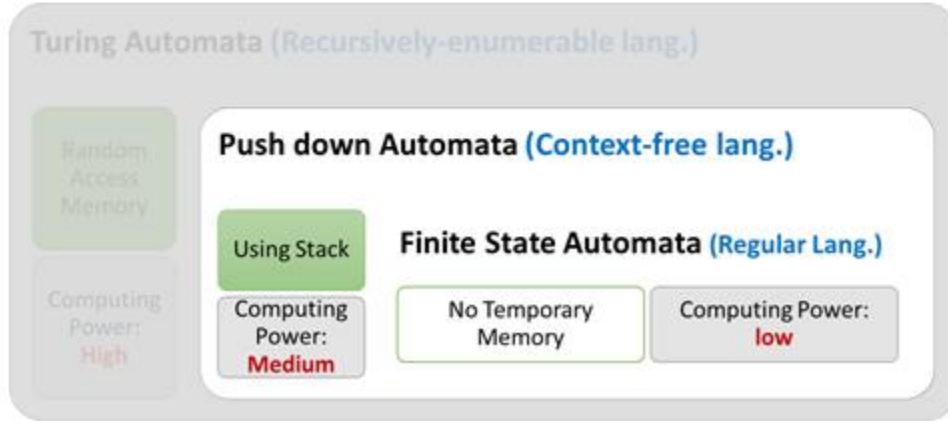
Chapter 2

Introduction

In chapter 2 :

- We present a **more powerful method** of describing languages:
  - called context-free grammars (CFG)

- the collection of **languages** associated with CFG are called the context-free languages (CFL).
  - They can describe languages with a recursive structure,

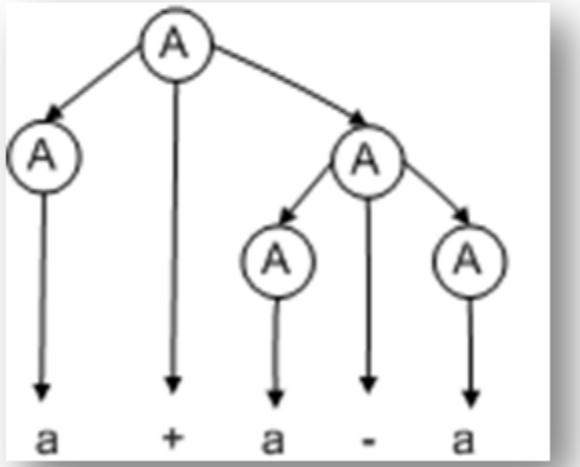


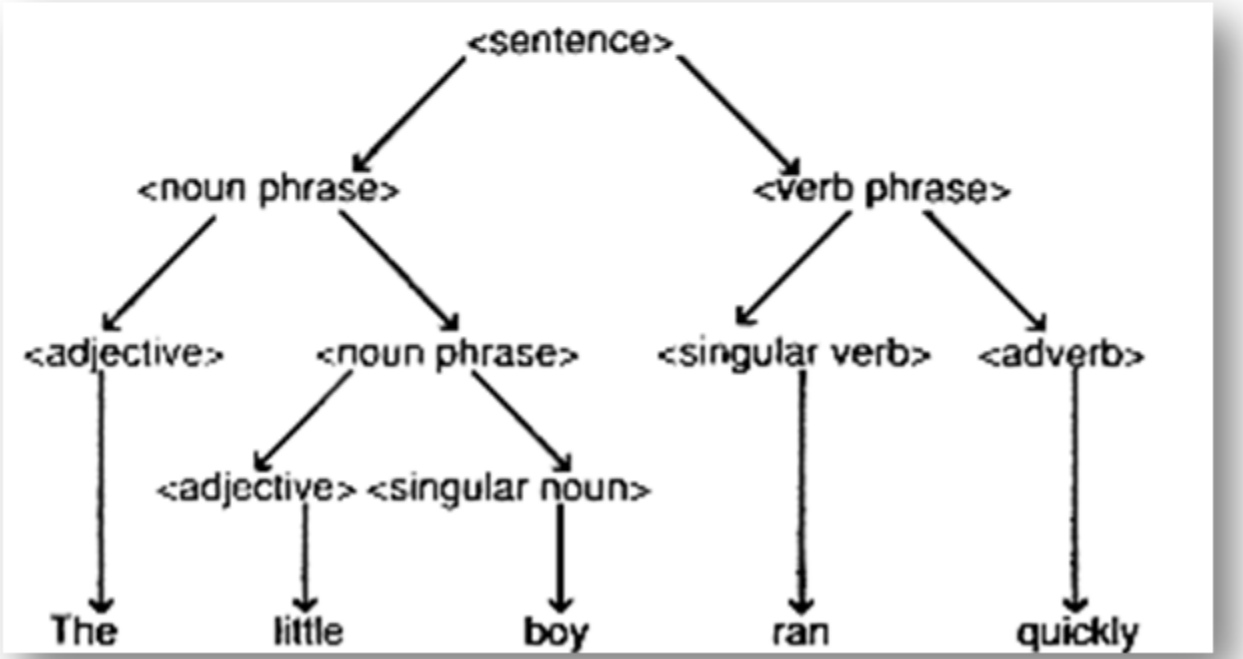
Introduction

context-free grammars:

- first **used** in the study of **human languages**
  - To organize and understand the **relationship of terms** such as noun, verb, and preposition and their respective phrases.

- another **application**: **compilers** and **interpreters** for programming languages using **parser**

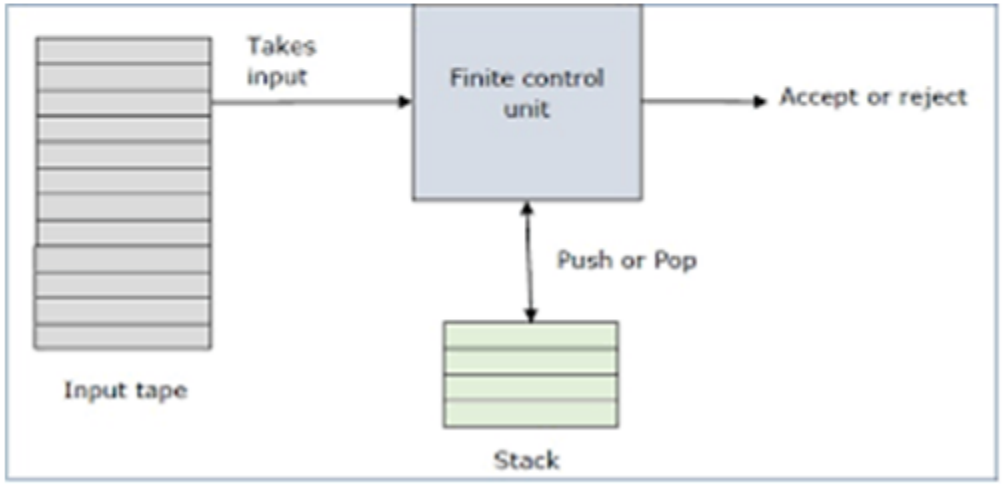




Introduction

pushdown automata:

- We also introduce pushdown automata, a class of machines recognizing the context-free languages



Chapter 2

2.1 Context-Free Grammars

CONTEXT-FREE GRAMMARS

Example : G_1 is a CFG.

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

CONTEXT-FREE GRAMMARS

Example : G_1 is a CFG.

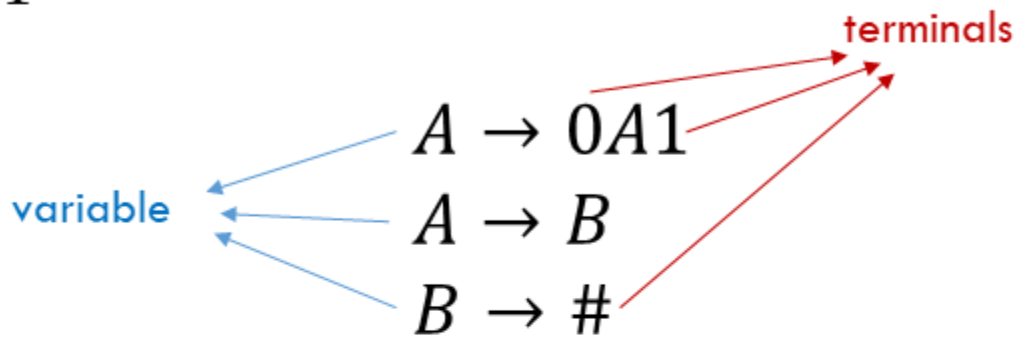
substitution rules (productions)

$$\left.\begin{array}{l} A \to 0A1 \\ A \to B \\ B \to \# \end{array}\right\} \quad A \to 0A1 \mid B$$

- A grammar consists of a collection of substitution rules, also called productions.
  - each rule appears as a line in the grammar

CONTEXT-FREE GRAMMARS

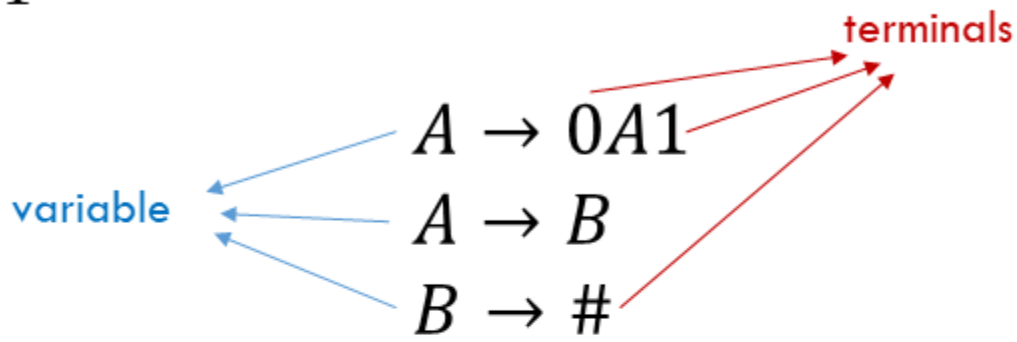
Example : G_1 is a CFG.



- A grammar consists of a collection of substitution rules, also called productions.
  - each rule appears as a line in the grammar
  - comprising a symbol (called a variable) and a string separated by an arrow. The **string** consists of variables and other symbols called terminals.

CONTEXT-FREE GRAMMARS

Example : G_1 is a CFG.



- A grammar consists of a collection of substitution rules, also called productions.
  - each rule appears as a line in the grammar
  - comprising a symbol (called a variable) and a string separated by an arrow. The **string** consists of variables and other symbols called terminals.
- **variable symbols:** often are represented by capital letters.
- **terminals** : are analogous to the input alphabet and often are represented by lowercase letters, numbers, or special symbols.

CONTEXT-FREE GRAMMARS

Example 1 : G_1 is a CFG.

start variable ← $A \rightarrow 0A1$

$A \rightarrow B$

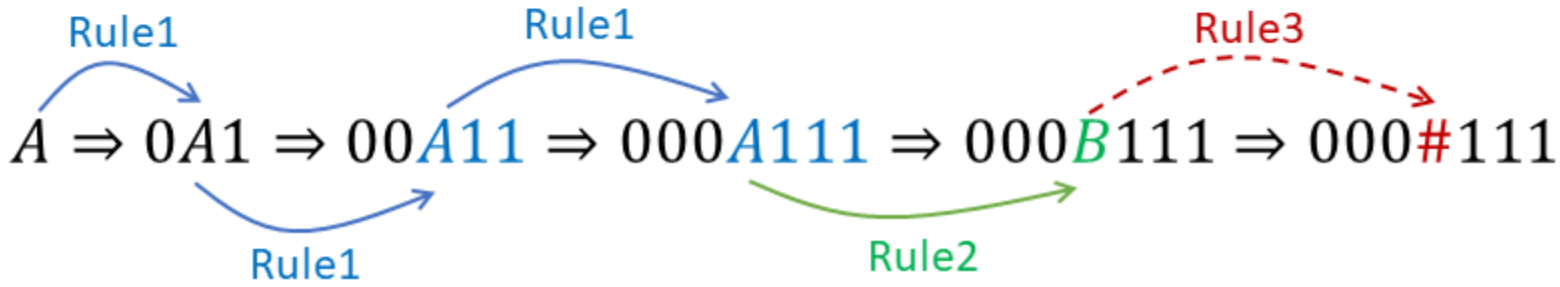
$B \rightarrow \#$

- A grammar consists of a collection of substitution rules, also called productions.
  - each rule appears as a line in the grammar
  - comprising a symbol (called a variable) and a string separated by an arrow. The **string** consists of variables and other symbols called terminals.
- **variable symbols:** often are represented by capital letters.
- **terminals** : are analogous to the input alphabet and often are represented by lowercase letters, numbers, or special symbols.
- One variable is designated as the start variable.
  - It usually occurs on the left-hand side of the topmost rule.

DERIVATION

A **grammar** describes a language by generating each string of that language in the following manner.

1. Write down the **start variable**.
2. Find a variable that is written down and a rule that starts with that variable. Replace the written down variable with the right-hand side of that rule.
3. Repeat step 2 until no variables remain.



G1

$A \rightarrow 0A1$

$A \rightarrow B$

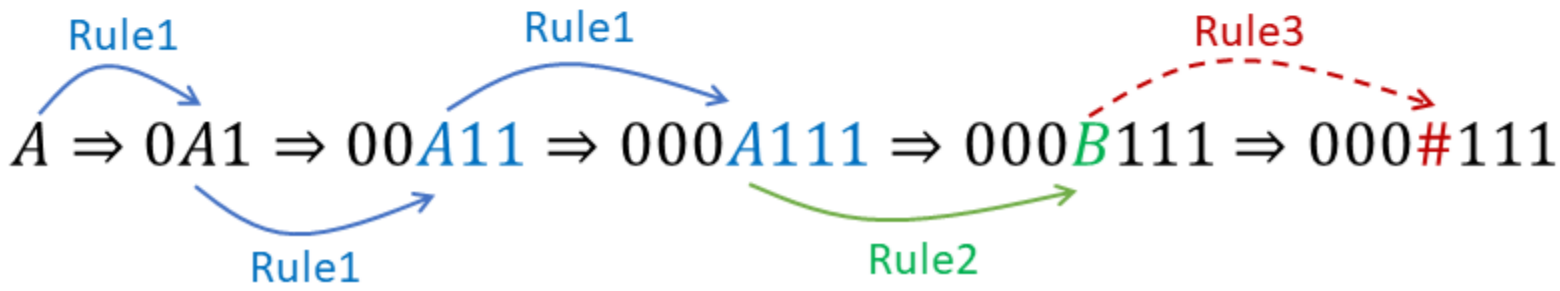
$B \rightarrow \#$

DERIVATION

A **grammar** describes a language by generating each string of that language in the following manner.

1. Write down the **start variable**.
2. Find a variable that is written down and a rule that starts with that variable. Replace the written down variable with the right-hand side of that rule.
3. Repeat step 2 until no variables remain.

- The **sequence of substitutions** to obtain a string is called a derivation.
- A derivation of string 000#111 in grammar G1 is



G1

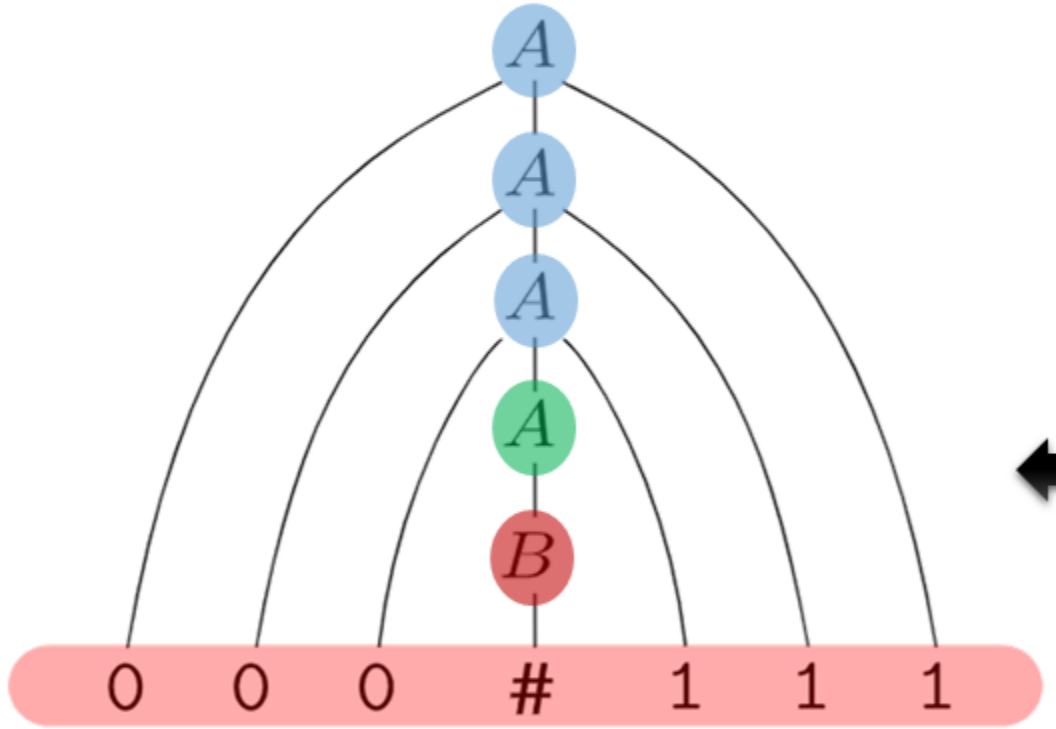
$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

PARSE TREE

You may also represent the same information pictorially with a parse tree.



G1

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

Rule1 Rule1 Rule3

$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$

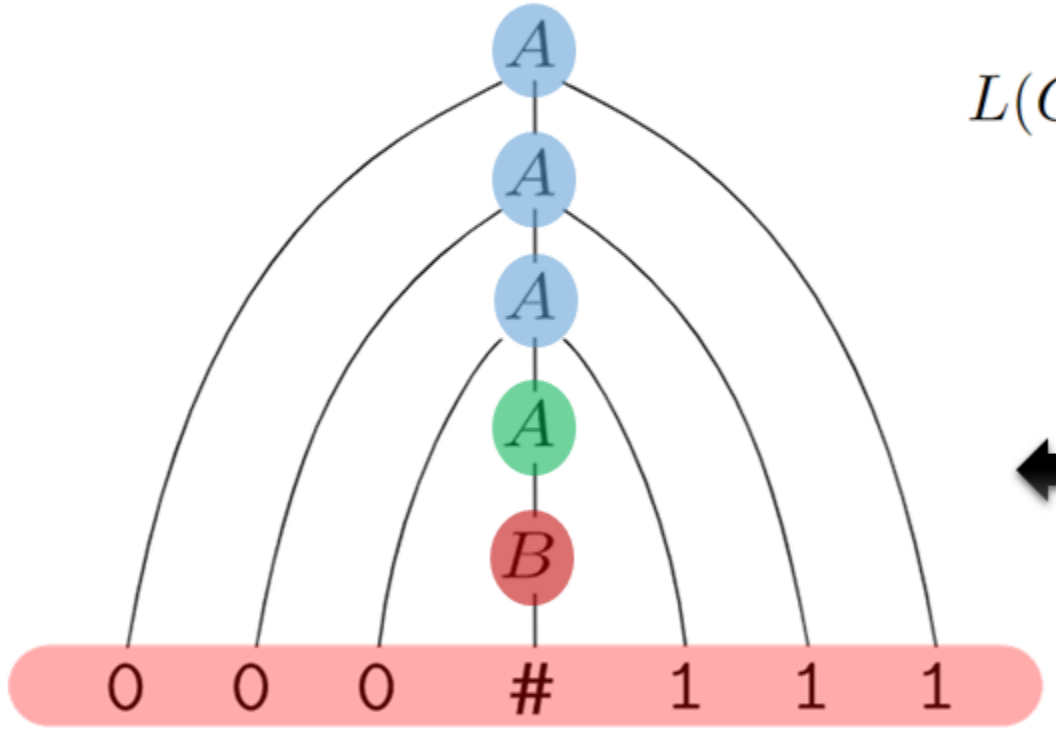
Rule1 Rule2

PARSE TREE

You may also represent the same information pictorially with a parse tree.

Example Sentences

- A
- 0A1
- 00A11
- 000A111
- 0000A1111
- 00000A11111
- 000000A111111
- 0000000A1111111
- 00000000A11111111
- 000000000A111111111



$L(G_1)$ is $\{0^n\#1^n \mid n \geq 0\}$

G1

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

Rule1 Rule1 Rule3

$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$

Rule1 Rule2

DERIVATION

Example 2 : G_2 which describes a fragment of the English language.

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
\langle \text{NOUN-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-NOUN} \rangle \mid \langle \text{CMPLX-NOUN} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{VERB-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-VERB} \rangle \mid \langle \text{CMPLX-VERB} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{PREP-PHRASE} \rangle &\rightarrow \langle \text{PREP} \rangle \langle \text{CMPLX-NOUN} \rangle \\
\langle \text{CMPLX-NOUN} \rangle &\rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \\
\langle \text{CMPLX-VERB} \rangle &\rightarrow \langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN-PHRASE} \rangle \\
\langle \text{ARTICLE} \rangle &\rightarrow \texttt{a} \mid \texttt{the} \\
\langle \text{NOUN} \rangle &\rightarrow \texttt{boy} \mid \texttt{girl} \mid \texttt{flower} \\
\langle \text{VERB} \rangle &\rightarrow \texttt{touches} \mid \texttt{likes} \mid \texttt{sees} \\
\langle \text{PREP} \rangle &\rightarrow \texttt{with}
\end{aligned}
$$

DERIVATION

Example 2 : G_2 which describes a fragment of the English language.

10 variables

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
\langle \text{NOUN-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-NOUN} \rangle \mid \langle \text{CMPLX-NOUN} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{VERB-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-VERB} \rangle \mid \langle \text{CMPLX-VERB} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{PREP-PHRASE} \rangle &\rightarrow \langle \text{PREP} \rangle \langle \text{CMPLX-NOUN} \rangle \\
\langle \text{CMPLX-NOUN} \rangle &\rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \\
\langle \text{CMPLX-VERB} \rangle &\rightarrow \langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN-PHRASE} \rangle \\
\langle \text{ARTICLE} \rangle &\rightarrow \texttt{a} \mid \texttt{the} \\
\langle \text{NOUN} \rangle &\rightarrow \texttt{boy} \mid \texttt{girl} \mid \texttt{flower} \\
\langle \text{VERB} \rangle &\rightarrow \texttt{touches} \mid \texttt{likes} \mid \texttt{sees} \\
\langle \text{PREP} \rangle &\rightarrow \texttt{with}
\end{aligned}
$$

- 27 terminals
- the standard **English alphabet** plus a **space character**.

18 rules

DERIVATION

Example 2 : G_2 which describes a fragment of the English language.

- Strings in L(G2) include:
  - a boy sees
  - the boy sees a flower
  - a girl with a flower likes the boy

⟨SENTENCE⟩ → ⟨NOUN-PHRASE⟩⟨VERB-PHRASE⟩
⟨NOUN-PHRASE⟩ → ⟨CMPLX-NOUN⟩ | ⟨CMPLX-NOUN⟩⟨PREP-PHRASE⟩
⟨VERB-PHRASE⟩ → ⟨CMPLX-VERB⟩ | ⟨CMPLX-VERB⟩⟨PREP-PHRASE⟩
⟨PREP-PHRASE⟩ → ⟨PREP⟩⟨CMPLX-NOUN⟩
⟨CMPLX-NOUN⟩ → ⟨ARTICLE⟩⟨NOUN⟩
⟨CMPLX-VERB⟩ → ⟨VERB⟩ | ⟨VERB⟩⟨NOUN-PHRASE⟩
⟨ARTICLE⟩ → a | the
⟨NOUN⟩ → boy | girl | flower
⟨VERB⟩ → touches | likes | sees
⟨PREP⟩ → with

$CFG\ G_2$

DERIVATION

Example 2 : G_2 which describes a fragment of the English language.

- Strings in L(G2) include:
  - a boy sees
  - the boy sees a flower
  - a girl with a flower likes the boy

(a derivation)

$$\begin{aligned}
&\langle \text{SENTENCE} \rangle \\
&\Rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{CMPLX-NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a}\ \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{CMPLX-VERB} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{VERB} \rangle \\
&\Rightarrow \texttt{a boy sees}
\end{aligned}$$

$$\begin{aligned}
\langle \text{SENTENCE} \rangle &\rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
\langle \text{NOUN-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-NOUN} \rangle \mid \langle \text{CMPLX-NOUN} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{VERB-PHRASE} \rangle &\rightarrow \langle \text{CMPLX-VERB} \rangle \mid \langle \text{CMPLX-VERB} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{PREP-PHRASE} \rangle &\rightarrow \langle \text{PREP} \rangle \langle \text{CMPLX-NOUN} \rangle \\
\langle \text{CMPLX-NOUN} \rangle &\rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \\
\langle \text{CMPLX-VERB} \rangle &\rightarrow \langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN-PHRASE} \rangle \\
\langle \text{ARTICLE} \rangle &\rightarrow \texttt{a} \mid \texttt{the} \\
\langle \text{NOUN} \rangle &\rightarrow \texttt{boy} \mid \texttt{girl} \mid \texttt{flower} \\
\langle \text{VERB} \rangle &\rightarrow \texttt{touches} \mid \texttt{likes} \mid \texttt{sees} \\
\langle \text{PREP} \rangle &\rightarrow \texttt{with}
\end{aligned}$$

$CFG\ G_2$

FORMAL DEFINITION OF A CONTEXT-FREE GRAMMAR

A ***context-free grammar*** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

G1

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

In grammar G1

- $V = \{A, B\}$
- $\Sigma = \{0, 1, \#\}$
- $S = A$

DERIVATION

- If $u, v,$ and w are strings of **variables and terminals**, and $A \rightarrow w$ is a rule of the grammar, we say that uAv yields uwv ,written $uAv \Rightarrow uwv$.

DERIVATION

- If $u, v,$ and w are strings of **variables and terminals**, and $A \to w$ is a rule of the grammar, we say that uAv yields uwv ,written $uAv \Rightarrow uwv$.

- Say that u derives v, written $u \overset{*}{\Rightarrow} v$,
  - if $u = v$ or
  - if a sequence $u_1, u_2, \ldots, u_k$ exists for k≥0 and

$$u \Rightarrow u1 \Rightarrow u2 \Rightarrow \cdots \Rightarrow uk \Rightarrow v.$$

- The **language of the grammar** is $\{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}$.

EXAMPLES OF CONTEXT-FREE GRAMMARS

- Consider grammar $G_3 = (\{S\}, \{a, b\}, R, S)$.
- The set of rules, R, is $S \rightarrow aSb|SS|\varepsilon$.

EXAMPLES OF CONTEXT-FREE GRAMMARS

- Consider grammar $G_3 = (\{S\}, \{a, b\}, R, S)$.
- The set of rules, R, is $S \rightarrow aSb|SS|\varepsilon$.

- This grammar generates :
  - strings such as $abab$, $aaabbb$, and $aababb$.

EXAMPLES OF CONTEXT-FREE GRAMMARS

- Consider grammar $G_3 = (\{S\}, \{a, b\}, R, S)$.
- The set of rules, R, is $S \rightarrow aSb|SS|\varepsilon$.

- This grammar generates :
  - strings such as $abab$, $aaabbb$, and $aababb$.
    - $S \Rightarrow SS \Rightarrow aSbS \Rightarrow aSbaSb \Rightarrow a\varepsilon baSb \Rightarrow aba\varepsilon b \Rightarrow abab$
    - $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaa\varepsilon bbb \Rightarrow aaabbb$
    - $S \Rightarrow aSb \Rightarrow aSSb \Rightarrow aaSbSb \Rightarrow aaSbaSbb \Rightarrow aa\varepsilon baSbb \Rightarrow aaba\varepsilon bb \Rightarrow aababb$

EXAMPLES OF CONTEXT-FREE GRAMMARS

- Consider grammar $G_3 = (\{S\}, \{a, b\}, R, S)$.
- The set of rules, R, is $S \rightarrow aSb|SS|\varepsilon$.

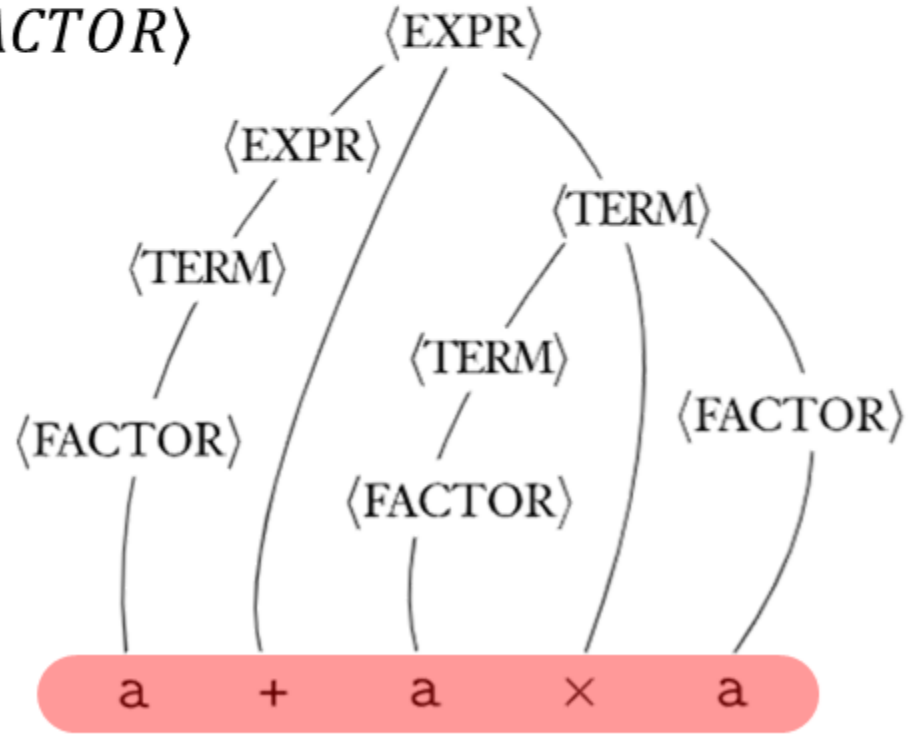
- This grammar generates :
  - strings such as $abab$, $aaabbb$, and $aababb$.
    - $S \Rightarrow SS \Rightarrow aSbS \Rightarrow aSbaSb \Rightarrow a\varepsilon ba\varepsilon b \Rightarrow abab$
    - $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaa\varepsilon bbb \Rightarrow aaabbb$
    - $S \Rightarrow aSb \Rightarrow aSSb \Rightarrow aaSbSb \Rightarrow aaSbaSbb \Rightarrow aa\varepsilon baSbb \Rightarrow aaba\varepsilon bb \Rightarrow aababb$

  - if you think of a as a left parenthes is "("and b as a right parenthes is ")"
    - Then L(G_3) is the language of all strings of properly nested parentheses.

EXAMPLES OF CONTEXT-FREE GRAMMARS

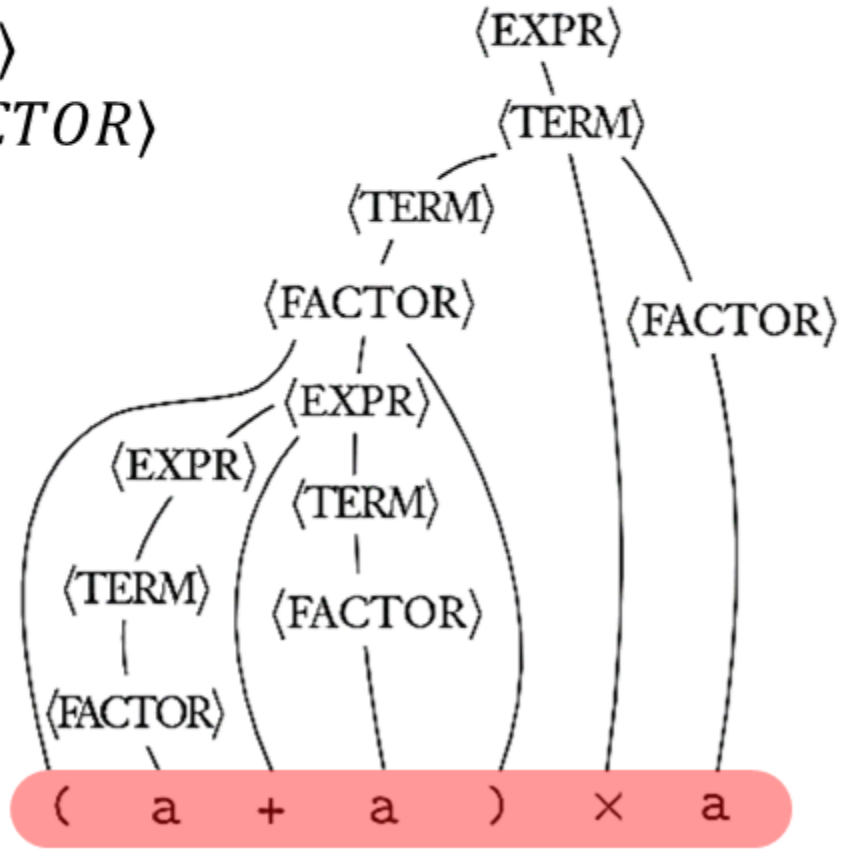
- Consider grammar $G_4 = (V, \Sigma, R, \langle EXPR \rangle)$
- $V = \{\langle EXPR \rangle, \langle TERM \rangle, \langle FACTOR \rangle\}$
- $\Sigma = \{a, +, \times, (,)\}$.
- The **rules** are
  - $\langle \boldsymbol{EXPR} \rangle \rightarrow \langle EXPR \rangle + \langle TERM \rangle | \langle TERM \rangle$
  - $\langle \boldsymbol{TERM} \rangle \rightarrow \langle TERM \rangle \times \langle FACTOR \rangle | \langle FACTOR \rangle$
  - $\langle \boldsymbol{FACTOR} \rangle \rightarrow (\langle EXPR \rangle) | a$

- **Grammar G_4** describes :
  - a fragment of a programming language concerned with arithmetic expressions.



EXAMPLES OF CONTEXT-FREE GRAMMARS

- Consider grammar $G_4 = (V, \Sigma, R, \langle EXPR \rangle)$
- $V = \{\langle EXPR \rangle, \langle TERM \rangle, \langle FACTOR \rangle\}$
- $\Sigma = \{a, +, \times, (,)\}$.
- The **rules** are
  - $\langle \boldsymbol{EXPR} \rangle \rightarrow \langle EXPR \rangle + \langle TERM \rangle | \langle TERM \rangle$
  - $\langle \boldsymbol{TERM} \rangle \rightarrow \langle TERM \rangle \times \langle FACTOR \rangle | \langle FACTOR \rangle$
  - $\langle \boldsymbol{FACTOR} \rangle \rightarrow (\langle EXPR \rangle) | a$



DESIGNING CONTEXT-FREE GRAMMARS

- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

DESIGNING CONTEXT-FREE GRAMMARS

- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 1: Based on this Fact: Many CFLs are the union of simpler CFLs.

DESIGNING CONTEXT-FREE GRAMMARS

- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 1: Based on this Fact: Many CFLs are the union of simpler CFLs.

- break a CFL into simpler pieces
- and then construct individual grammars for each piece
- merge the individual grammars into a grammar for the original language
  - By combining their rules and then adding the new rule

$$S \to S_1 | S_2 | \cdots | S_k,$$

Where the variables S_i are the start variables for the individual grammars.

DESIGNING CONTEXT-FREE GRAMMARS

- Example: Construct a grammar for the language

$$\{0^n 1^n | n \geq 0\} \cup \{1^n 0^n | n \geq 0\}.$$

DESIGNING CONTEXT-FREE GRAMMARS

- Example: Construct a grammar for the language

$$\{0^n 1^n | n \geq 0\} \cup \{1^n 0^n | n \geq 0\}.$$

- $L_1 = \{0^n 1^n | n \geq 0\}$ Rules: $S_1 \rightarrow 0S_1 1 | \varepsilon$

DESIGNING CONTEXT-FREE GRAMMARS

- Example: Construct a grammar for the language

$$\{0^n 1^n | n \geq 0\} \cup \{1^n 0^n | n \geq 0\}.$$

- $L_1 = \{0^n 1^n | n \geq 0\}$ Rules: $S_1 \rightarrow 0S_1 1 | \varepsilon$
- $L_2 = \{1^n 0^n | n \geq 0\}$ Rules: $S_2 \rightarrow 1S_2 0 | \varepsilon$

DESIGNING CONTEXT-FREE GRAMMARS

- Example: Construct a grammar for the language

$$\{0^n 1^n | n \geq 0\} \cup \{1^n 0^n | n \geq 0\}.$$

- $L_1 = \{0^n 1^n | n \geq 0\}$ Rules: $S_1 \rightarrow 0S_1 1 | \varepsilon$
- $L_2 = \{1^n 0^n | n \geq 0\}$ Rules: $S_2 \rightarrow 1S_2 0 | \varepsilon$

- $L_1 \cup L_2$ Rules: $S \rightarrow S1 | S2$
  $S_1 \rightarrow 0S_1 1 | \varepsilon$
  $S_2 \rightarrow 1S_2 0 | \varepsilon$

DESIGNING CONTEXT-FREE GRAMMARS

- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 2: constructing a CFG for a regular language :

DESIGNING CONTEXT-FREE GRAMMARS

- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 2: constructing a CFG for a regular language :

- construct a DFA for that language.

DESIGNING CONTEXT-FREE GRAMMARS

- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 2: constructing a CFG for a regular language :

- construct a DFA for that language.
- convert the DFA into an equivalent CFG as follows.

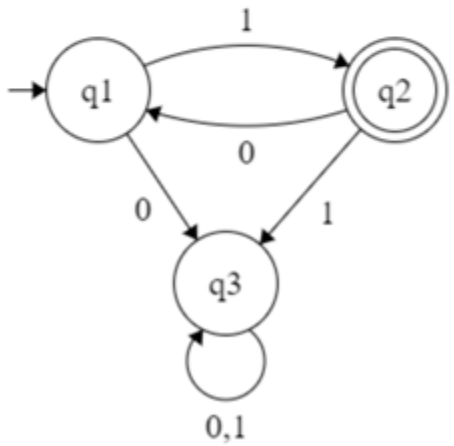
DESIGNING CONTEXT-FREE GRAMMARS

- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 2: constructing a CFG for a regular language :

- construct a DFA for that language.
- convert the DFA into an equivalent CFG as follows.



DESIGNING CONTEXT-FREE GRAMMARS

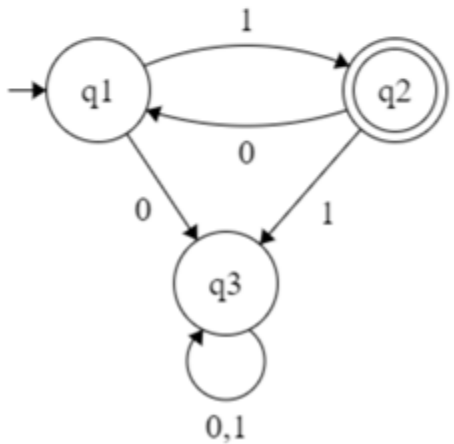
- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 2: constructing a CFG for a regular language :

- construct a DFA for that language.
- convert the DFA into an equivalent CFG as follows.
  - Make a variable R_i for each state q_i of the DFA.

- $V = \{R1, R2, R3\}$



DESIGNING CONTEXT-FREE GRAMMARS

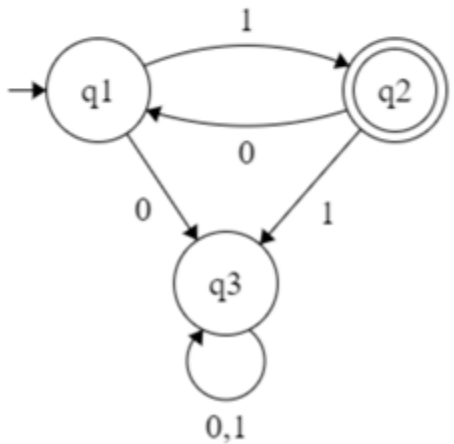
- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 2: constructing a CFG for a regular language :

- construct a DFA for that language.
- convert the DFA into an equivalent CFG as follows.
  - Make a variable R_i for each state q_i of the DFA.
  - Add the rule $R_i \rightarrow aR_j$ to the CFG if $\delta(q_i,a)=q_j$.

- $V = \{R1, R2, R3\}$
- $R_1 \rightarrow 0R_3 \mid 1R_2 \quad R_2 \rightarrow 0R_1 \mid 1R_3 \quad R_3 \rightarrow 1R_3 \mid 1R_3$



DESIGNING CONTEXT-FREE GRAMMARS

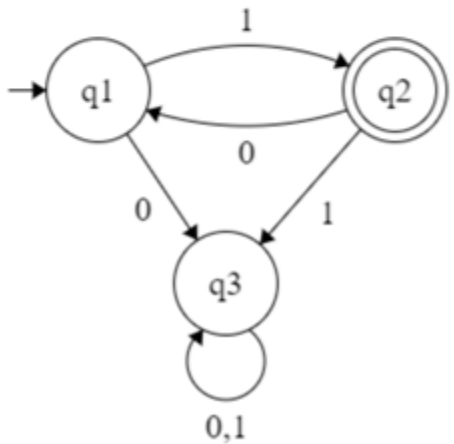
- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 2: constructing a CFG for a regular language :

- construct a DFA for that language.
- convert the DFA into an equivalent CFG as follows.
  - Make a variable R_i for each state q_i of the DFA.
  - Add the rule $R_i \rightarrow aR_j$ to the CFG if $\delta(q_i,a)=q_j$.
  - Add the rule $R_i \rightarrow \varepsilon$ if q_i is an accept state of the DFA.

- $V = \{R1, R2, R3\}$
- $\boldsymbol{R_1} \rightarrow 0R_3 \mid 1R_2 \quad R_2 \rightarrow 0R_1 \mid 1R_3 \quad R_3 \rightarrow 1R_3 \mid 1R_3$
- $R_2 \rightarrow \varepsilon$



DESIGNING CONTEXT-FREE GRAMMARS

- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 2: constructing a CFG for a regular language :

- construct a DFA for that language.
- convert the DFA into an equivalent CFG as follows.
  - Make a variable R_i for each state q_i of the DFA.
  - Add the rule $R_i \rightarrow aR_j$ to the CFG if $\delta(q_i,a)=q_j$.
  - Add the rule $R_i \rightarrow \varepsilon$ if q_i is an accept state of the DFA.
  - Make R_0 the start variable of the grammar, where q_0 is the start state of the machine.
  - the resulting CFG generates the same language that the DFA recognizes. (Verify it on your own)

DESIGNING CONTEXT-FREE GRAMMARS

- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 3: constructing a CFG for a Certain context-free languages :

DESIGNING CONTEXT-FREE GRAMMARS

- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 3: constructing a CFG for a Certain context-free languages :

- strings with two substrings that are "linked"
  - need to remember an unbounded amount of information about one of the substrings
    - to verify that it corresponds properly to the other substring.
  - **Example**: $\{0^n 1^n | n \geq 0\}$

DESIGNING CONTEXT-FREE GRAMMARS

- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 3: constructing a CFG for a Certain context-free languages :

- strings with two substrings that are "linked"
  - need to remember an unbounded amount of information about one of the substrings
    - to verify that it corresponds properly to the other substring.
  - **Example**: $\{0^n 1^n | n \geq 0\}$

- **construct a CFG:** by using a rule of the form $R \rightarrow uRv$

DESIGNING CONTEXT-FREE GRAMMARS

- The design of context-free grammars requires creativity.
- Trickier to construct than finite automata

Helpful techniques

Method 4: constructing a CFG for a language with a recursive structure:

- **Example**:

- Consider grammar $G_4 = (V, \Sigma, R, \langle EXPR \rangle)$
- $V = \{\langle EXPR \rangle, \langle TERM \rangle, \langle FACTOR \rangle\}$
- $\Sigma = \{a, +, \times, (,)\}$.
- The **rules** are
  - $\langle \boldsymbol{EXPR} \rangle \rightarrow \langle EXPR \rangle + \langle TERM \rangle | \langle TERM \rangle$
  - $\langle \boldsymbol{TERM} \rangle \rightarrow \langle TERM \rangle \times \langle FACTOR \rangle | \langle FACTOR \rangle$
  - $\langle \boldsymbol{FACTOR} \rangle \rightarrow (\langle EXPR \rangle) | a$

AMBIGUITY

- Sometimes a grammar can generate the same string in several different ways.
  - We say, the string is derived ambiguously in that grammar.

AMBIGUITY

- Sometimes a grammar can generate the same string in several different ways.
  - We say, the string is derived ambiguously in that grammar.

- If a grammar generates some string ambiguously,
  - then the grammar is ambiguous.

AMBIGUITY

- Sometimes a grammar can generate the same string in several different ways.
  - We say, the string is derived ambiguously in that grammar.

- If a grammar generates some string ambiguously,
  - then the grammar is ambiguous.

- Ambiguity is because of
  - several different parse trees (not two different derivations.)

AMBIGUITY

- Sometimes a grammar can generate the same string in several different ways.
  - We say, the string is derived ambiguously in that grammar.

- If a grammar generates some string ambiguously,
  - then the grammar is ambiguous.

- Ambiguity is because of
  - several different parse trees (not two different derivations.)
  - thus several different meanings
  - **undesirable** for certain applications, such as programming languages
    - They need a unique interpretation.

AMBIGUITY

Example: consider grammarG5:

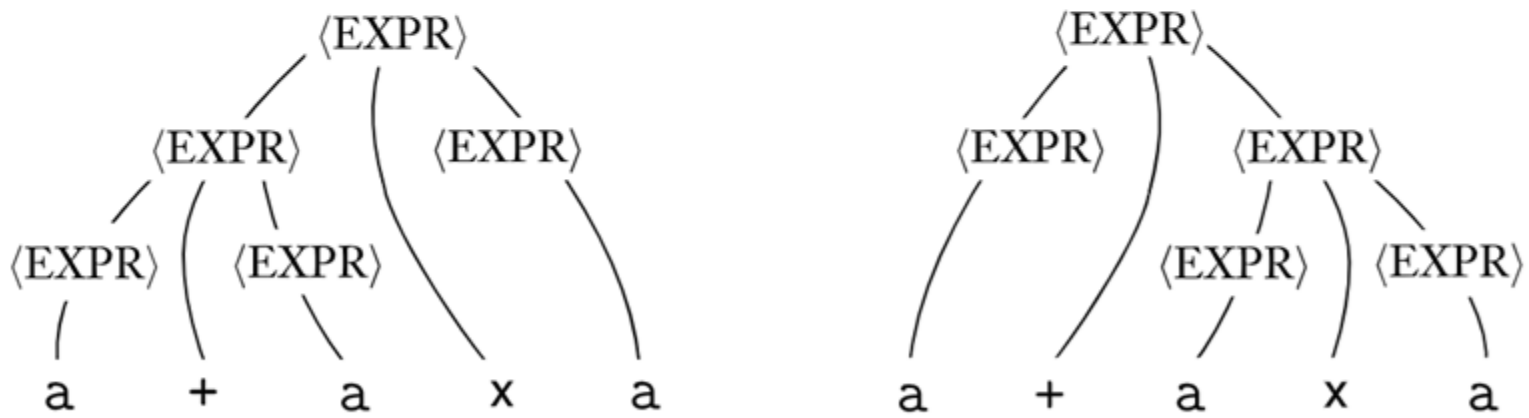
$$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle | \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle | (\langle \text{EXPR} \rangle) | \text{a}$$

AMBIGUITY

Example: consider grammarG5:

$$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle | \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle | (\langle \text{EXPR} \rangle) | \text{a}$$

This grammar generates the string a + a × a ambiguously.



LEFTMOST AND RIGHTMOST DERIVATION

- A derivation of a string w in a grammar G is a leftmost derivation
  - if **at every step** the leftmost remaining variable is the one replaced.

- A derivation of a string w in a grammar G is a rightmost derivation
  - if **at every step** the rightmost remaining variable is the one replaced.

LEFTMOST AND RIGHTMOST DERIVATION

Example: Let any set of production rules in a CFG be

$$X \rightarrow X + X \mid X \times X \mid |X| \; a$$

over an alphabet $\{a\}$. We want to generate $a + a \times a$.

LEFTMOST AND RIGHTMOST DERIVATION

Example: Let any set of production rules in a CFG be

$$X \rightarrow X + X \mid X \times X \mid |X| \; a$$

over an alphabet $\{a\}$. We want to generate $a + a \times a$.

- leftmost derivation :
  - $X \rightarrow X + X \rightarrow a + X \rightarrow a + X \times X \rightarrow a + a \times X \rightarrow a + a \times a$

LEFTMOST AND RIGHTMOST DERIVATION

Example: Let any set of production rules in a CFG be

$$X \rightarrow X + X \mid X \times X \mid |X| \; a$$

over an alphabet $\{a\}$. We want to generate $a + a \times a$.

- leftmost derivation :
  - $X \rightarrow X + X \rightarrow a + X \rightarrow a + X \times X \rightarrow a + a \times X \rightarrow a + a \times a$
- rightmost derivation
  - $X \rightarrow X \times X \rightarrow X \times a \rightarrow X + X \times a \rightarrow X + a \times a \rightarrow a + a \times a$

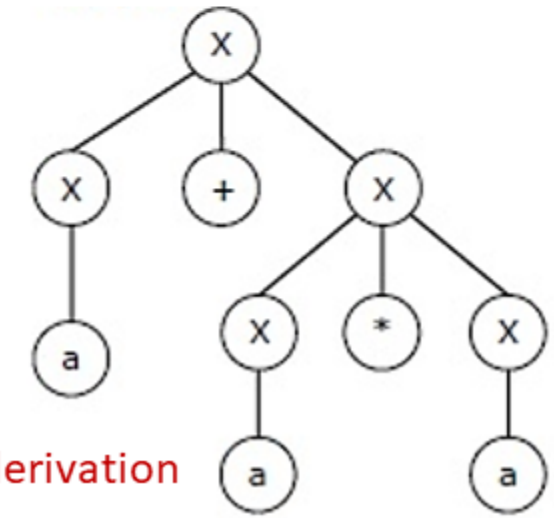
LEFTMOST AND RIGHTMOST DERIVATION

Example: Let any set of production rules in a CFG be

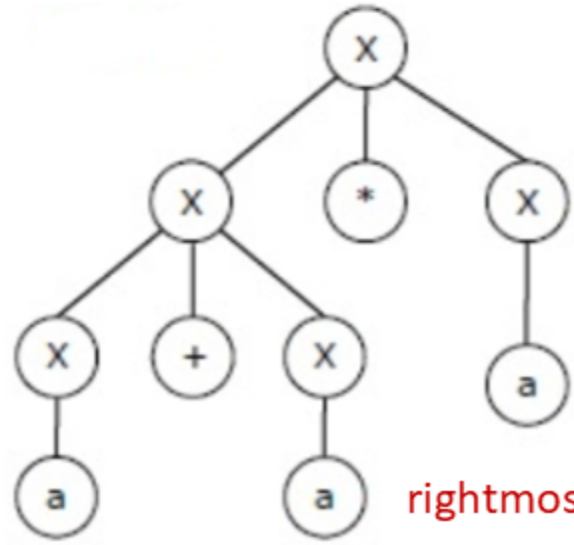
$$X \rightarrow X + X \mid X \times X \mid |X| \mid a$$

over an alphabet $\{a\}$. We want to generate $a + a \times a$.

- leftmost derivation :
  - $X \rightarrow X + X \rightarrow a + X \rightarrow a + X \times X \rightarrow a + a \times X \rightarrow a + a \times a$
- rightmost derivation
  - $X \rightarrow X \times X \rightarrow X \times a \rightarrow X + X \times a \rightarrow X + a \times a \rightarrow a + a \times a$



leftmost derivation



rightmost derivation

AMBIGUITY AND LEFTMOST AND RIGHTMOST DERIVATION

- A string w is derived ambiguously in context-free grammar G if it has two or more different leftmost derivations.

- Grammar G is ambiguous if it generates some string ambiguously.

AMBIGUITY AND LEFTMOST AND RIGHTMOST DERIVATION

- A string *w* is derived ambiguously in context-free grammar G if it has two or more different leftmost derivations.
- Grammar G is ambiguous if it generates some string ambiguously.
- Some context-free languages, can be generated
  - **using an ambiguous grammar and an unambiguous grammar**.
- Some context-free languages, however, can be generated
  - only by ambiguous grammars.
  - Such languages are called inherently ambiguous.

Parse tree and deviation

- For every parse tree, there is a unique leftmost, and a unique rightmost derivation.

- The pars tree of an unambiguous grammar is unique.
  - every generated string in a CFG has a unique parse tree.

CHOMSKY NORMAL FORM

- One of the simplest and most useful forms of context-free grammars is called Chomsky normal form.

CHOMSKY NORMAL FORM

- One of the simplest and most useful forms of context-free grammars is called Chomsky normal form.

A context-free grammar is in ***Chomsky normal form*** if every rule is of the form

$$A \to BC$$
$$A \to a$$

where a is any terminal and A, B, and C are any variables—except that B and C may not be the start variable. In addition, we permit the rule $S \to \varepsilon$, where S is the start variable.

CHOMSKY NORMAL FORM

- Theorem: Any context-free language is generated by a context-free grammar in Chomsky normal form.

CHOMSKY NORMAL FORM

Theorem: Any context-free language is generated by a context-free grammar in Chomsky normal form.

PROOF:

1. **Eliminate the start symbol from right-hand sides**
   - we add a new start variable S_0 and a new rule $S_0 \rightarrow S$,
   - where S was the original start variable.

CHOMSKY NORMAL FORM

- Theorem: Any context-free language is generated by a context-free grammar in Chomsky normal form.

PROOF:

1. **Eliminate the start symbol from right-hand sides**
   - we add a new start variable S_0 and a new rule $S_0 \rightarrow S$,
   - where S was the original start variable.

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

$$S0 \rightarrow S$$
$$S \rightarrow ASA \mid aB$$
$$A \rightarrow B \mid S$$
$$B \rightarrow b \mid \varepsilon$$

CHOMSKY NORMAL FORM

$$A \rightarrow BC$$
$$A \rightarrow a$$

CHOMSKY NORMAL FORM

PROOF: (cont.)

2. Removing ε-rules.

- We remove an ε-rule $A \to \varepsilon$, where ***A* is not the start variable**.
- Then for each occurrence of an A on the right-hand side of a rule,
  - we add a new rule with that occurrence deleted.
  - In other words, if $R \to uAv$ is a rule in which u and v are strings of variables and terminals, we add rule $R \to uv$.

$$
\begin{aligned}
S0 &\to S \\
S &\to ASA|aB \\
A &\to B|S \\
B &\to b|\varepsilon
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
S0 &\to S \\
S &\to ASA|aB|a \\
A &\to B|S|\varepsilon \\
B &\to b|\varepsilon
\end{aligned}
$$

CHOMSKY NORMAL FORM

PROOF: (cont.)

2. **Removing ε-rules.** (cont.)
   - We do so **for each occurrence of an A**,
   - **Example**:
     - $A \to \varepsilon,\ R \to uAvAw \quad \Rightarrow\ R \to uvAw \mid uAvw \mid uvw.$
     - $A \to \varepsilon,\ R \to A \quad \Rightarrow\ R \to A \mid \varepsilon$ (unless we had previously removed it)

   - We **repeat these steps** until we eliminate all ε-rules not involving the start variable.

CHOMSKY NORMAL FORM

PROOF: (cont.)

3. we handle all unit rules.

- We remove a unit rule $A \to B$.
- Then, whenever a rule $B \to u$ appears,
  - we add the rule $A \to u$ unless this was a unit rule previously removed.
  - As before, u is a string of variables and terminals.
- We repeat these steps until we eliminate all unit rules.

$$\begin{matrix} A \to B \\ B \to CDEF \end{matrix} \quad \Rightarrow \quad A \to CDEF$$

CHOMSKY NORMAL FORM

PROOF: (cont.)

4. **Finally, we convert all remaining rules into the proper form.**
   - (Eliminate right-hand sides with more than 2 nonterminals)

CHOMSKY NORMAL FORM

PROOF: (cont.)

4. Finally, we convert all remaining rules into the proper form.

- (Eliminate right-hand sides with more than 2 nonterminals)
- We replace each rule $A \to u_1 u_2 \cdots u_k$,
  - where k≥3 and each u_i is a variable or terminal symbol,
- with the rules $A \to u_1 A_1$, $A_1 \to u_2 A_2$, $A_2 \to u_3 A_3$,...,and $A_{k-2} \to u_{k-1} u_k$.
  - The A_i's are new variables.

CHOMSKY NORMAL FORM

PROOF: (cont.)

4. Finally, we convert all remaining rules into the proper form.

- (Eliminate right-hand sides with more than 2 nonterminals)
- We replace each rule $A \rightarrow u_1 u_2 \cdots u_k$,
  - where k≥3 and each u_i is a variable or terminal symbol,
- with the rules $A \rightarrow u_1 A_1$, $A_1 \rightarrow u_2 A_2$, $A_2 \rightarrow u_3 A_3$,...,and $A_{k-2} \rightarrow u_{k-1} u_k$.
  - The A_i's are new variables.
- We replace any terminal u_i in the preceding rule(s) with the new variable U_i and add the rule $U_i \rightarrow u_i$.

CHOMSKY NORMAL FORM

PROOF: (cont.)

4. Finally, we convert all remaining rules into the proper form.

- (Eliminate right-hand sides with more than 2 nonterminals)
- We replace each rule $A \to u_1 u_2 \cdots u_k$,
  - where k≥3 and each u_i is a variable or terminal symbol,
- with the rules $A \to u_1 A_1$, $A_1 \to u_2 A_2$, $A_2 \to u_3 A_3$,…,and $A_{k-2} \to u_{k-1} u_k$.
  - The A_i's are new variables.
- We replace any terminal u_i in the preceding rule(s) with the new variable U_i and add the rule $U_i \to u_i$.

$$S0 \to ASA|aB|a|SA|AS$$
$$S \to ASA|aB|a|SA|AS$$
$$A \to S|b|ASA|aB|a|SA|AS$$
$$B \to b$$

$$S0 \to AA_1|UB|a|SA|AS$$
$$S \to AA_1|UB|a|SA|AS$$
$$A \to b|AA_1|UB|a|SA|AS$$
$$A_1 \to SA$$
$$U \to a$$
$$B \to b$$

CHOMSKY NORMAL FORM

Example: Convert the following CFG to CNF

$$S \rightarrow ASA|aB$$
$$A \rightarrow B|S$$
$$B \rightarrow b|\varepsilon$$

- **Step1:** Eliminate the start symbol from right-hand sides and add **a new start symbol**

$$S0 \rightarrow S$$
$$S \rightarrow ASA|aB$$
$$A \rightarrow B|S$$
$$B \rightarrow b|\varepsilon$$

CHOMSKY NORMAL FORM

Example: (Cont.)

- **Step2:** Eliminate all ε-rules

$$S0 \rightarrow S$$
$$S \rightarrow ASA|aB$$
$$A \rightarrow B|S$$
$$B \rightarrow b|\varepsilon$$

Previous rules

$$S0 \rightarrow S$$
$$S \rightarrow ASA|aB|a$$
$$A \rightarrow B|S|\varepsilon$$
$$B \rightarrow b|\varepsilon$$

$$S0 \rightarrow S$$
$$S \rightarrow ASA|aB|a|SA|AS|S$$
$$A \rightarrow B|S|\varepsilon$$
$$B \rightarrow b$$

CHOMSKY NORMAL FORM

Example: (Cont.)

- **Step3a:** Eliminate all unit rules; $S \rightarrow S$ and $S0 \rightarrow S$

$S0 \rightarrow S$
$S \rightarrow ASA|aB|a|SA|AS|S$
$A \rightarrow B|S|\varepsilon$
$B \rightarrow b$

Previous rules

$S0 \rightarrow S$
$S \rightarrow ASA|aB|a|SA|AS|S$
$A \rightarrow B|S$
$B \rightarrow b$

$S0 \rightarrow ASA|aB|a|SA|AS$
$S \rightarrow ASA|aB|a|SA|AS$
$A \rightarrow B|S$
$B \rightarrow b$

CHOMSKY NORMAL FORM

Example: (Cont.)

- **Step3b:** Eliminate all unit rules; $A \rightarrow B$ and $A \rightarrow S$

$$S0 \rightarrow S|ASA|aB|a|SA|AS$$
$$S \rightarrow ASA|aB|a|SA|AS$$
$$A \rightarrow B|S$$
$$B \rightarrow b$$

Previous rules

$$S0 \rightarrow ASA|aB|a|SA|AS$$
$$S \rightarrow ASA|aB|a|SA|AS$$
$$A \rightarrow B|S|b$$
$$B \rightarrow b$$

$$S0 \rightarrow ASA|aB|a|SA|AS$$
$$S \rightarrow ASA|aB|a|SA|AS$$
$$A \rightarrow S|b|ASA|aB|a|SA|AS$$
$$B \rightarrow b$$

CHOMSKY NORMAL FORM

Example: (Cont.)

- **Step4:** Convert the remaining rules into the proper form by adding additional variables and rules.

$$S0 \rightarrow ASA|aB|a|SA|AS$$
$$S \rightarrow ASA|aB|a|SA|AS$$
$$A \rightarrow S|b|ASA|aB|a|SA|AS$$
$$B \rightarrow b$$

Previous rules

$$S0 \rightarrow AA_1|UB|a|SA|AS$$
$$S \rightarrow AA_1|UB|a|SA|AS$$
$$A \rightarrow b|AA_1|UB|a|SA|AS$$
$$A_1 \rightarrow SA$$
$$U \rightarrow a$$
$$B \rightarrow b$$

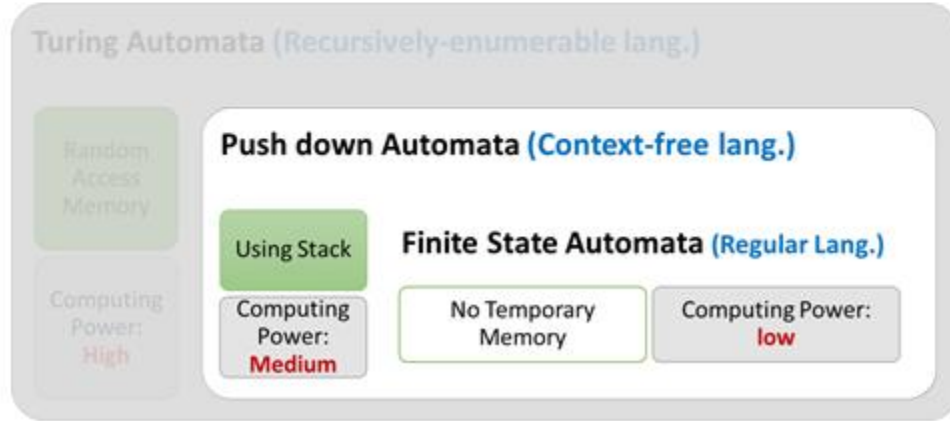
$$A \rightarrow BC$$
$$A \rightarrow a$$

Chapter 2

2.2 Pushdown Automata

PUSHDOWN AUTOMATA (PDA)

- PDAs are like nondeterministic finite automata but have an extra component called a stack.
  - provides additional memory beyond the finite amount available in the control.
  - allows pushdown automata to recognize some nonregular languages.



Chapter 2

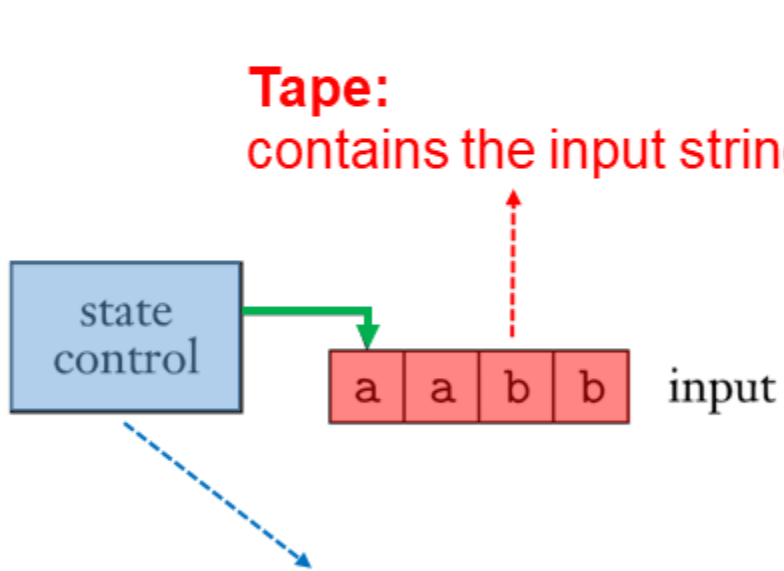
PUSHDOWN AUTOMATA (PDA)

- Pushdown automata are equivalent in power to context-free grammars.

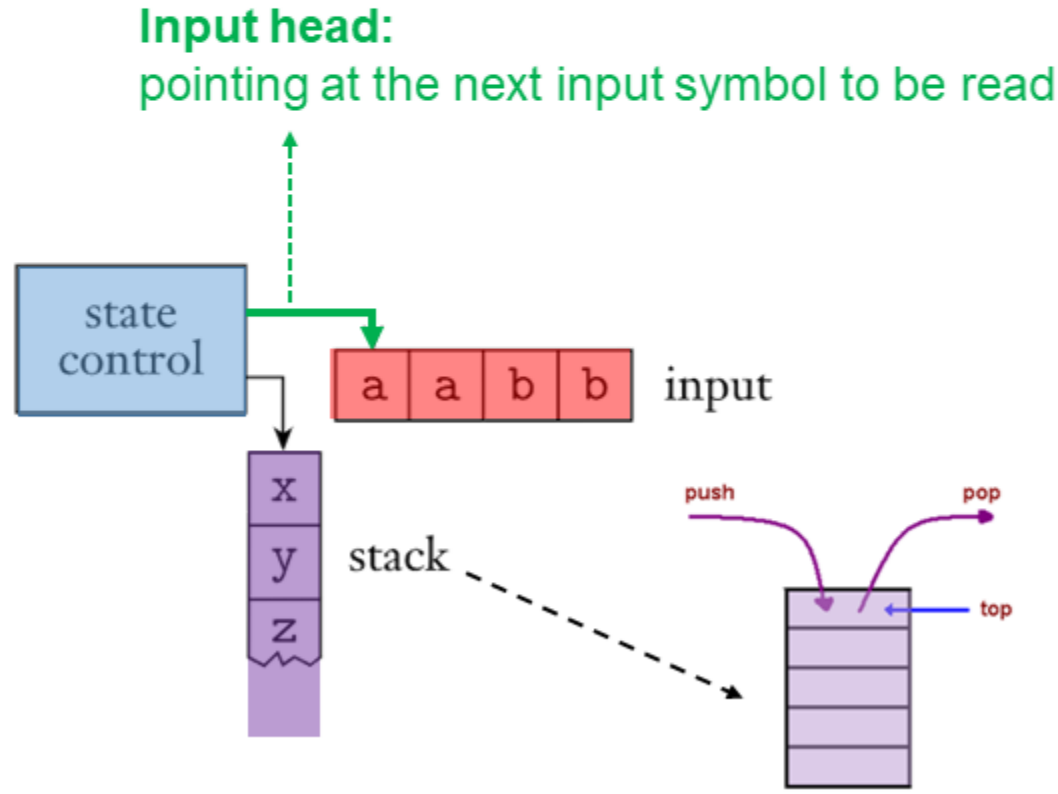
- A language is context free if
  - a **context-free grammar** generating it or
  - a **push-down automaton** recognizing it.

PUSHDOWN AUTOMATA (PDA)

- A PDA can write symbols on the stack and read them back later.
  - Pushing : writing a symbol on the stack
  - Popping : removing a symbol from the stack.
  - stack is a "last in, first out" storage device.



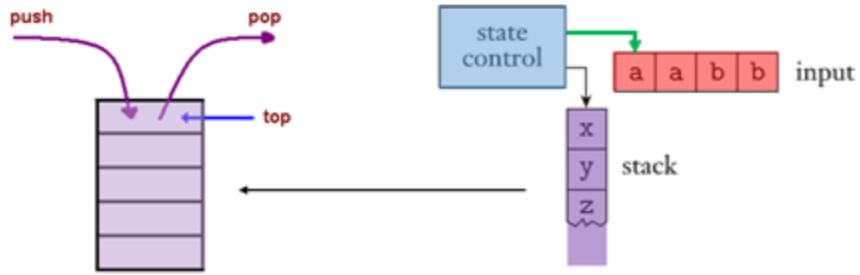
Finite automata



Pushdown automata

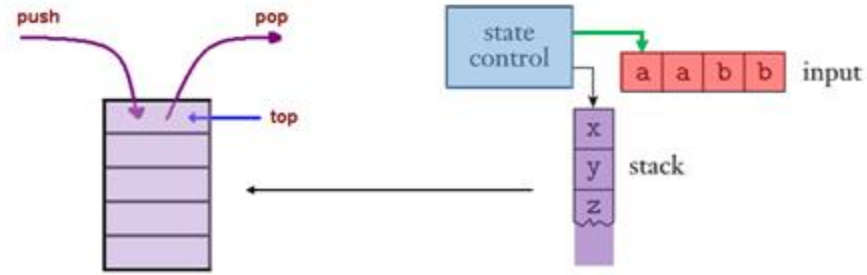
PUSHDOWN AUTOMATA (PDA)

- Stack is of infinite size.
- Stack is a "last in, first out" storage device.
- **Example:** How a PDA recognizes the nonregular language $\{0^n 1^n | n \geq 0\}$.



PUSHDOWN AUTOMATA (PDA)

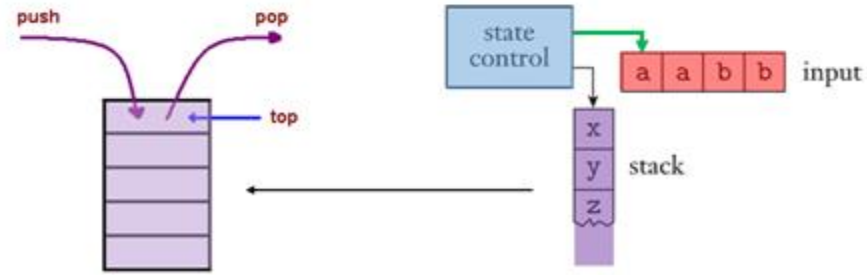
- Stack is of infinite size.
- Stack is a "last in, first out" storage device.
- **Example:** How a PDA recognizes the nonregular language $\{0^n 1^n | n \geq 0\}$.
- Can use its stack to store the number of 0's it has seen.



PUSHDOWN AUTOMATA (PDA)

- Stack is of infinite size.
- Stack is a "last in, first out" storage device.
- **Example:** How a PDA recognizes the nonregular language $\{0^n 1^n | n \geq 0\}$.
- Can use its stack to store the number of 0's it has seen.

> - As each 0 is read, push it onto the stack
> - As soon as 1's are read, pop a 0 off the stack
> - If reading the input is finished exactly when the stack is empty, accept the input else reject the input



DETERMINISTIC AND NONDETERMINISTIC (PDA)

- **Deterministic** and **nondeterministic** pushdown automata are not equivalent in power.
  - Nondeterministic pushdown automata recognize certain languages that no deterministic pushdown automata can recognize.

- Note: nondeterministic PDAs are equivalent in power to context-free grammars.

FORMAL DEFINITION OF A PUSHDOWN AUTOMATON

A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

Where $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$

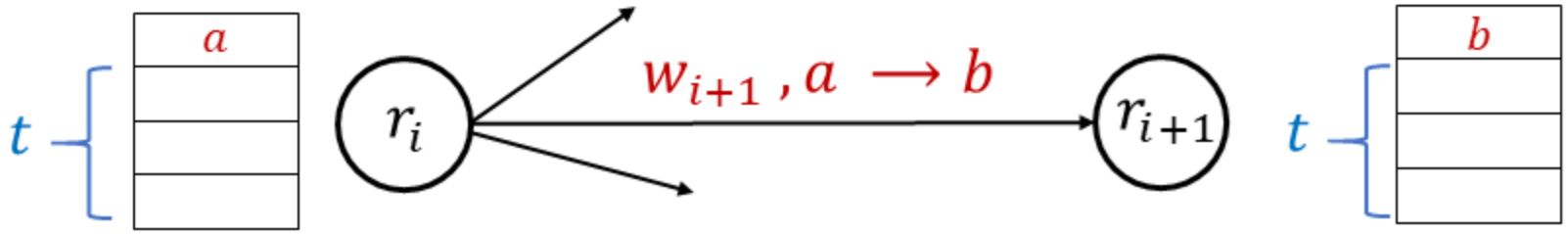
COMPUTAIONS OF A PUSHDOWN AUTOMATON

- A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows.
- It accepts input $w = w_1 w_2 \cdots w_m$, where each $w_i \in \Sigma_\varepsilon$ and sequences of states $r_0, r_1, \cdots, r_m \in Q$ and strings $s_0, s_1, \cdots, s_m \in \Gamma^*$ exist that satisfy the following conditions.

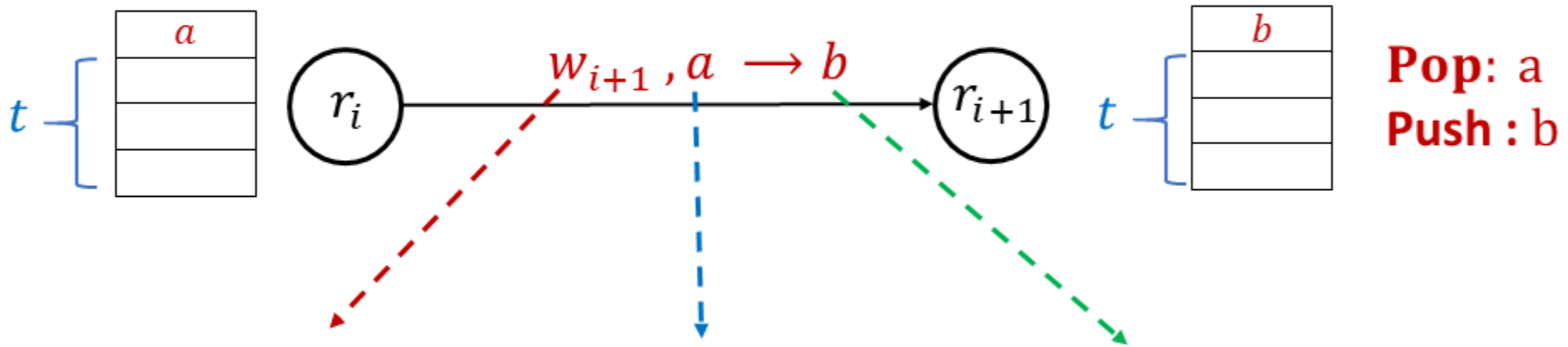
1. $r_0 = q_0$ and $s_0 = \varepsilon$.

2. For $i = 0, \ldots, m-1$, we 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 $t \in \Gamma^*$. This condition states that M moves properly according to the state, stack, and next input symbol.

3. $r_m \in F$. This condition states that an accept state occurs at the input end.



COMPUTAIONS OF A PUSHDOWN AUTOMATON



Input symbol
Maybe ε

Symbol on top of the stack.
This symbol is popped.
Maybe ε : means the stack is
Neither read nor popped

This symbol is pushed onto the stack.
This symbol is popped.
Maybe ε : means nothing is pushed.

EXAMPLES OF PUSHDOWN AUTOMATA

- **Example 1:** The formal description of the PDA that recognizes the language $\{0^n 1^n | n \geq 0\}$.

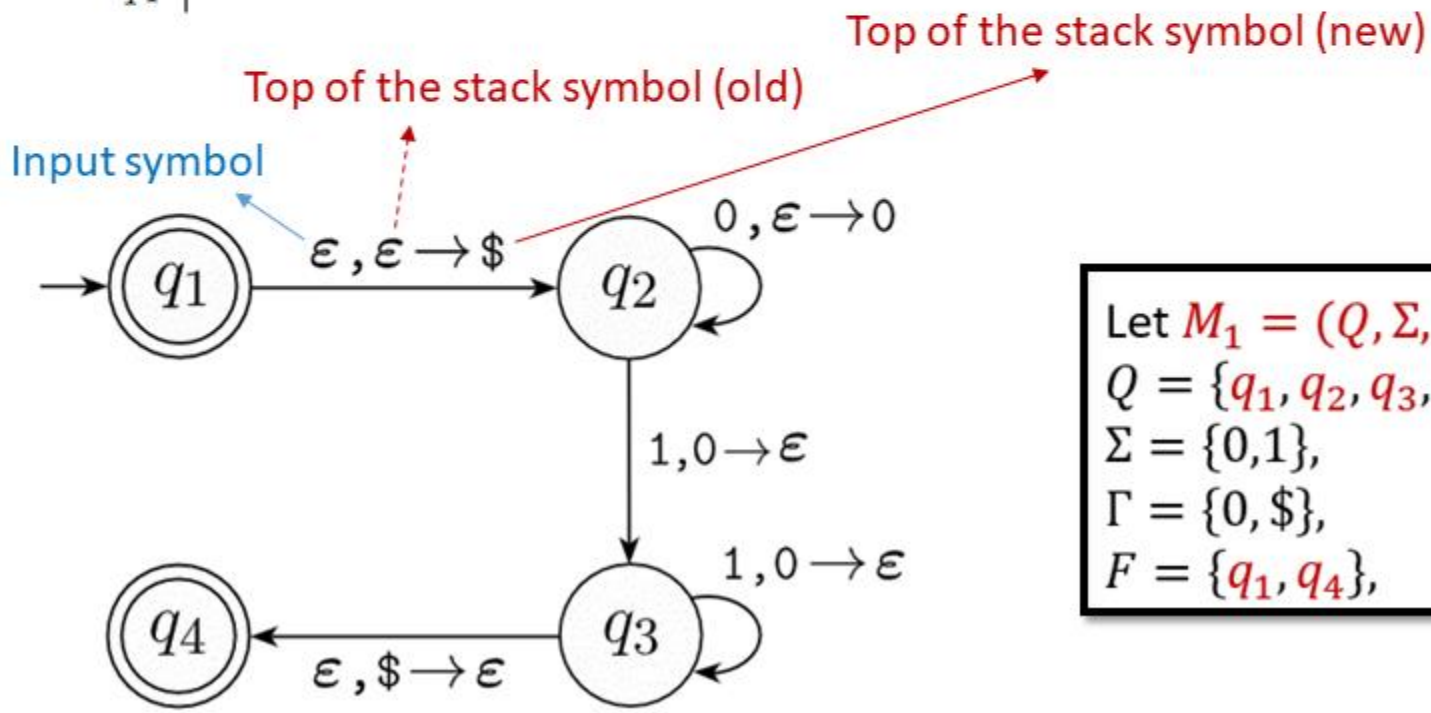
- Let $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, F)$, where
- $Q = \{q_1, q_2, q_3, q_4\}$,
- $\Sigma = \{0,1\}$,
- $\Gamma = \{0, \$\}$,
- $F = \{q_1, q_4\}$,
- And δ is given by the following table, wherein blank entries signify Ø.

Input:	0			1			ε		
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									

EXAMPLES OF PUSHDOWN AUTOMATA

- **Example 1:** (cont.)

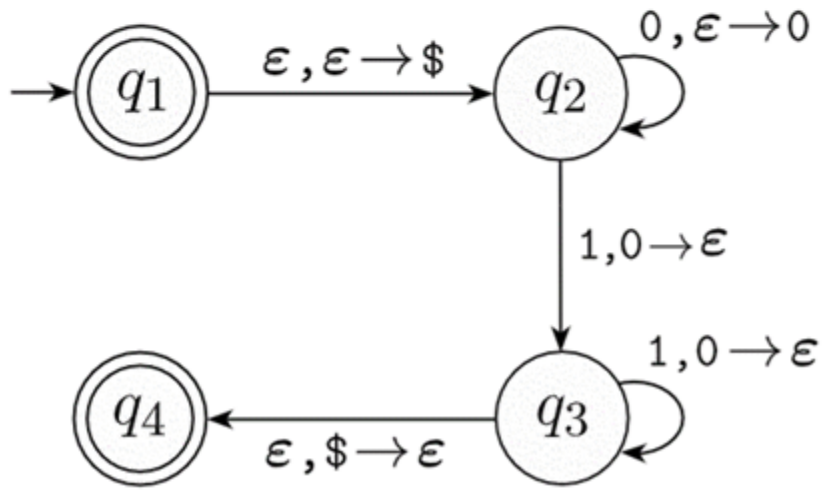
Input:	0			1			ε		
Stack:	0	$	ε	0	$	ε	0	$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									



Let $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, F)$, where
$Q = \{q_1, q_2, q_3, q_4\}$,
$\Sigma = \{0,1\}$,
$\Gamma = \{0, \$\}$,
$F = \{q_1, q_4\}$,

EXAMPLES OF PUSHDOWN AUTOMATA

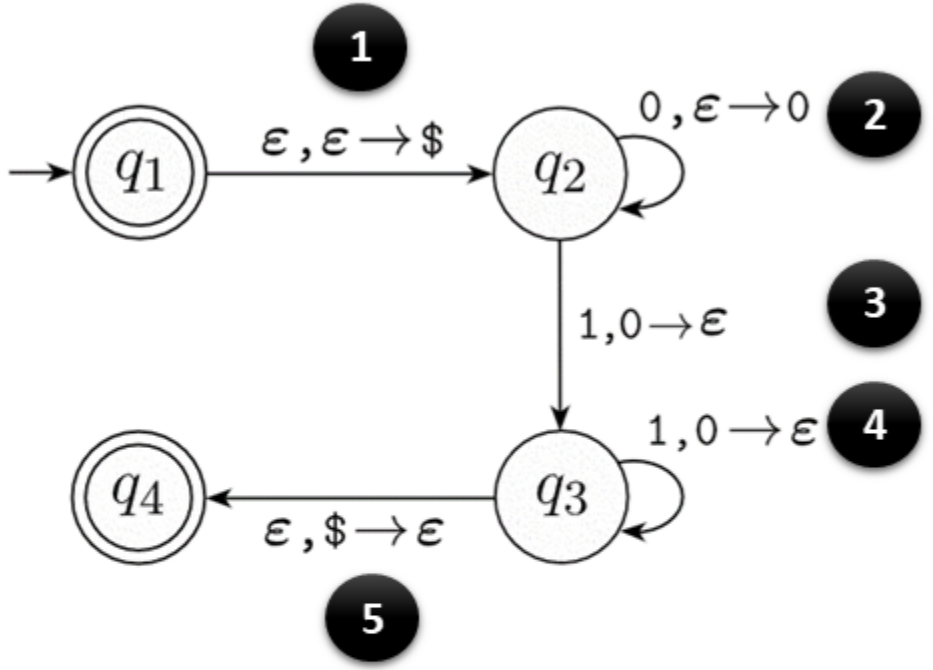
- **Example 1:** (cont.)
- "$a, b \rightarrow c$" : to signify that when the machine is reading an a from the input, it may replace the symbol b on the top of the stack with a c.
- $a = \varepsilon$: the machine may make this transition without reading any symbol from the input.
- $b = \varepsilon$: the machine may make this transition without reading and popping any symbol from the stack.
- $c = \varepsilon$: the machine does not write any symbol on the stack when going along this transition.



EXAMPLES OF PUSHDOWN AUTOMATA

- **Example 1:** (cont.)

- no explicit mechanism to allow the PDA to test for an empty stack.
- This PDA is able to get the same effect by initially placing a special **symbol $** on the stack.



- $b = \varepsilon$: the machine may make this transition without
the st

- $c =$
on the

1. Creating an empty stack
2. If input is 0, then push a 0 onto the stack
3,4. If input is 1, then pop a 0 from the stack and no push
5. The last input symbol and stack is empty → accept the string

EXAMPLES OF PUSHDOWN AUTOMATA

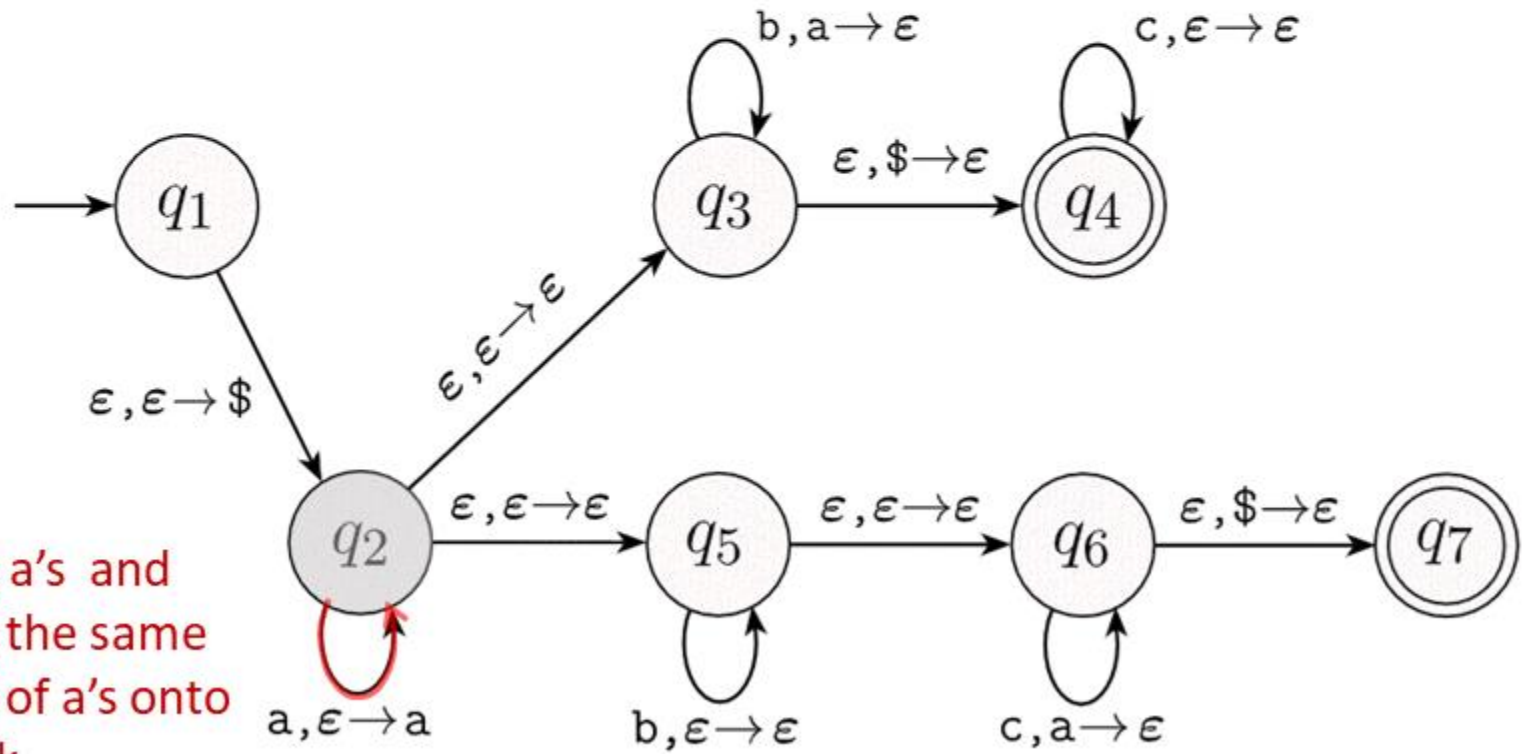
- **Example 3:** Find the PDA that recognizes the language

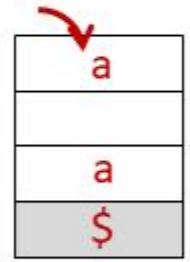
$$\{\mathrm{a}^i\mathrm{b}^j\mathrm{c}^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}.$$

EXAMPLES OF PUSHDOWN AUTOMATA

- **Example 2:** Find the PDA that recognizes the language

$$\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}.$$



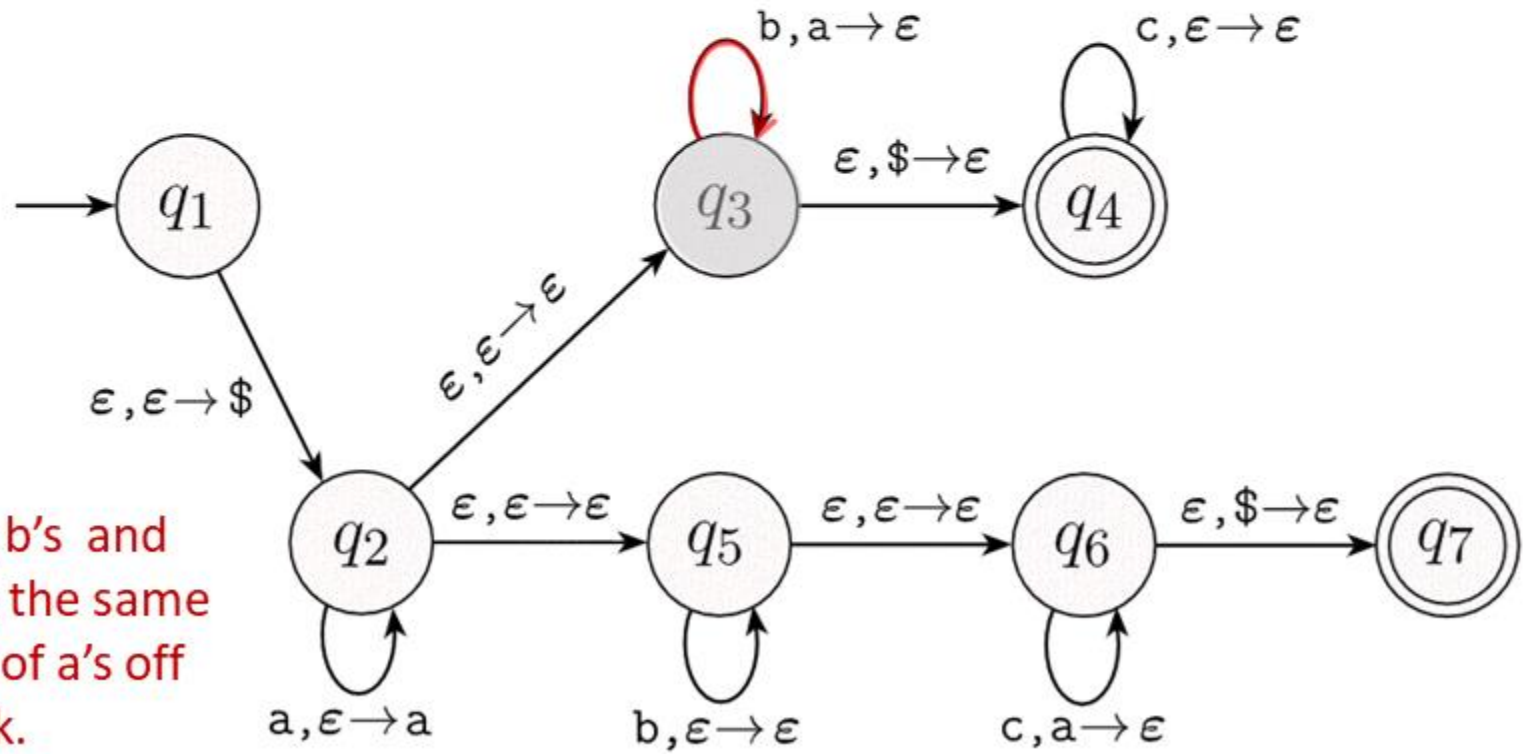


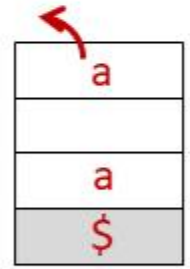
- Reading a's and pushing the same amount of a's onto the stack.

EXAMPLES OF PUSHDOWN AUTOMATA

- **Example 2:** Find the PDA that recognizes the language

$$\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}.$$



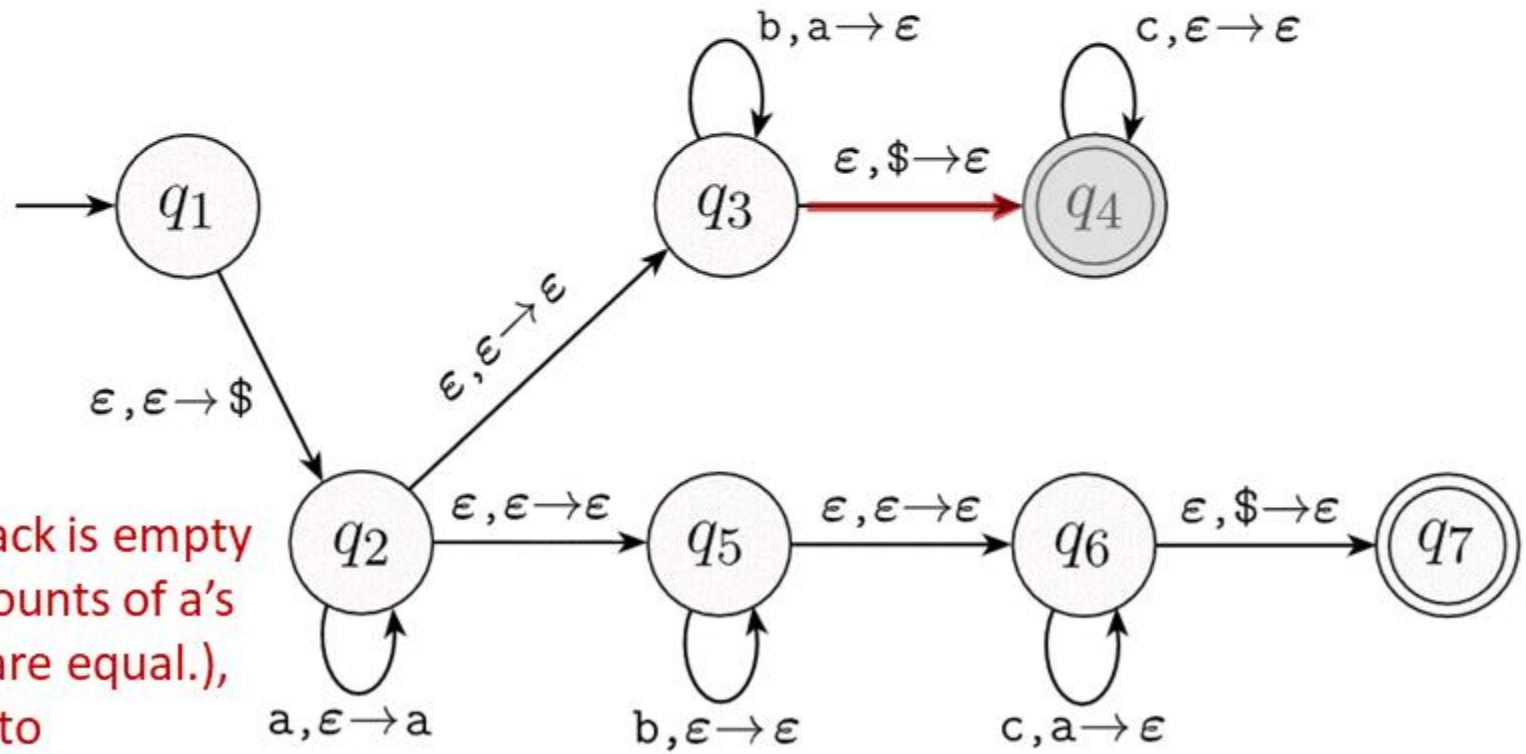


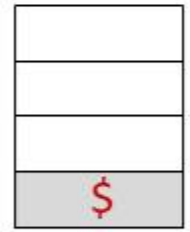
- Reading b's and popping the same amount of a's off the stack.

EXAMPLES OF PUSHDOWN AUTOMATA

- **Example 2:** Find the PDA that recognizes the language

$$\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}.$$



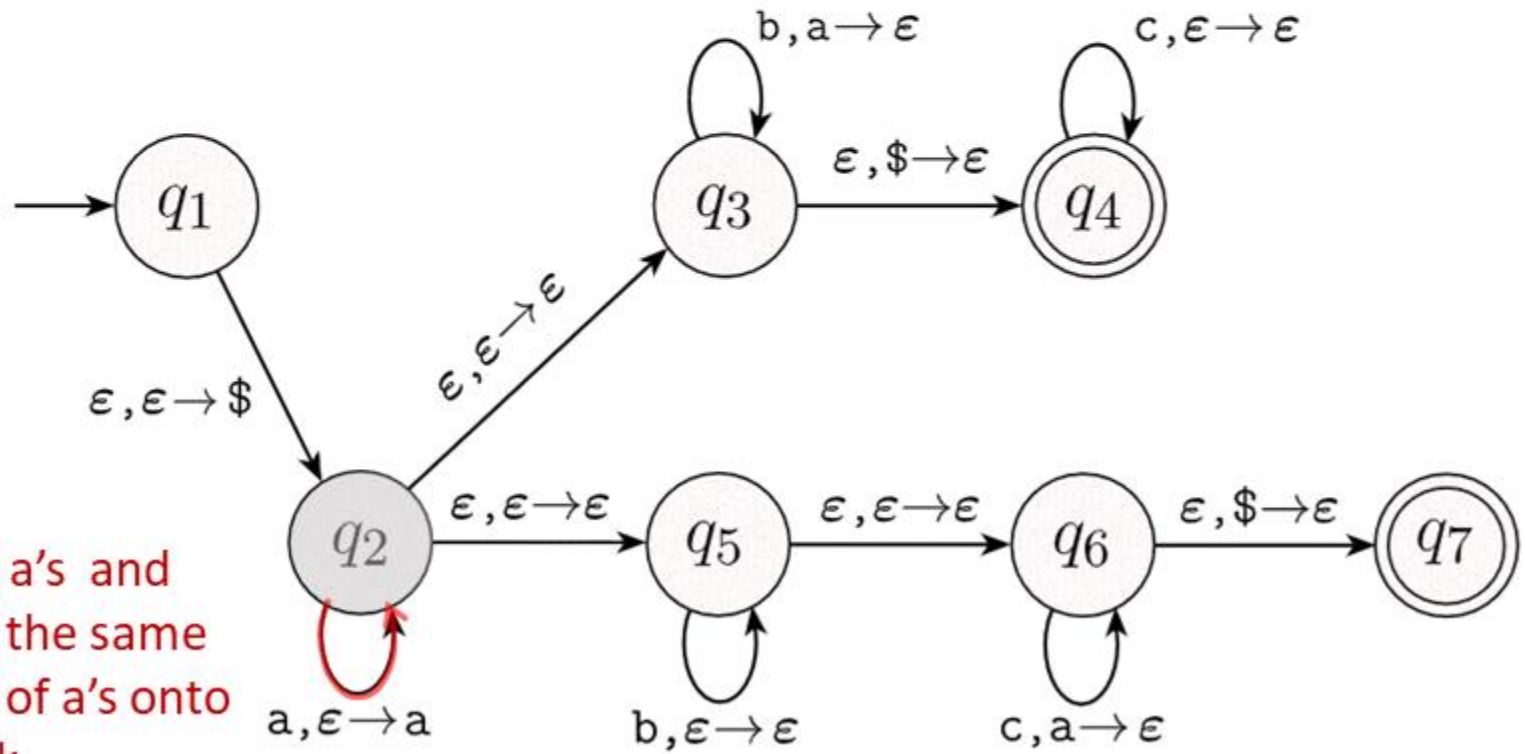


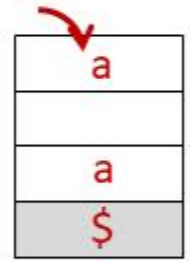
- If the stack is empty (the amounts of a's and b's are equal.), then go to the accept state.

EXAMPLES OF PUSHDOWN AUTOMATA

- **Example 2:** Find the PDA that recognizes the language

$$\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}.$$



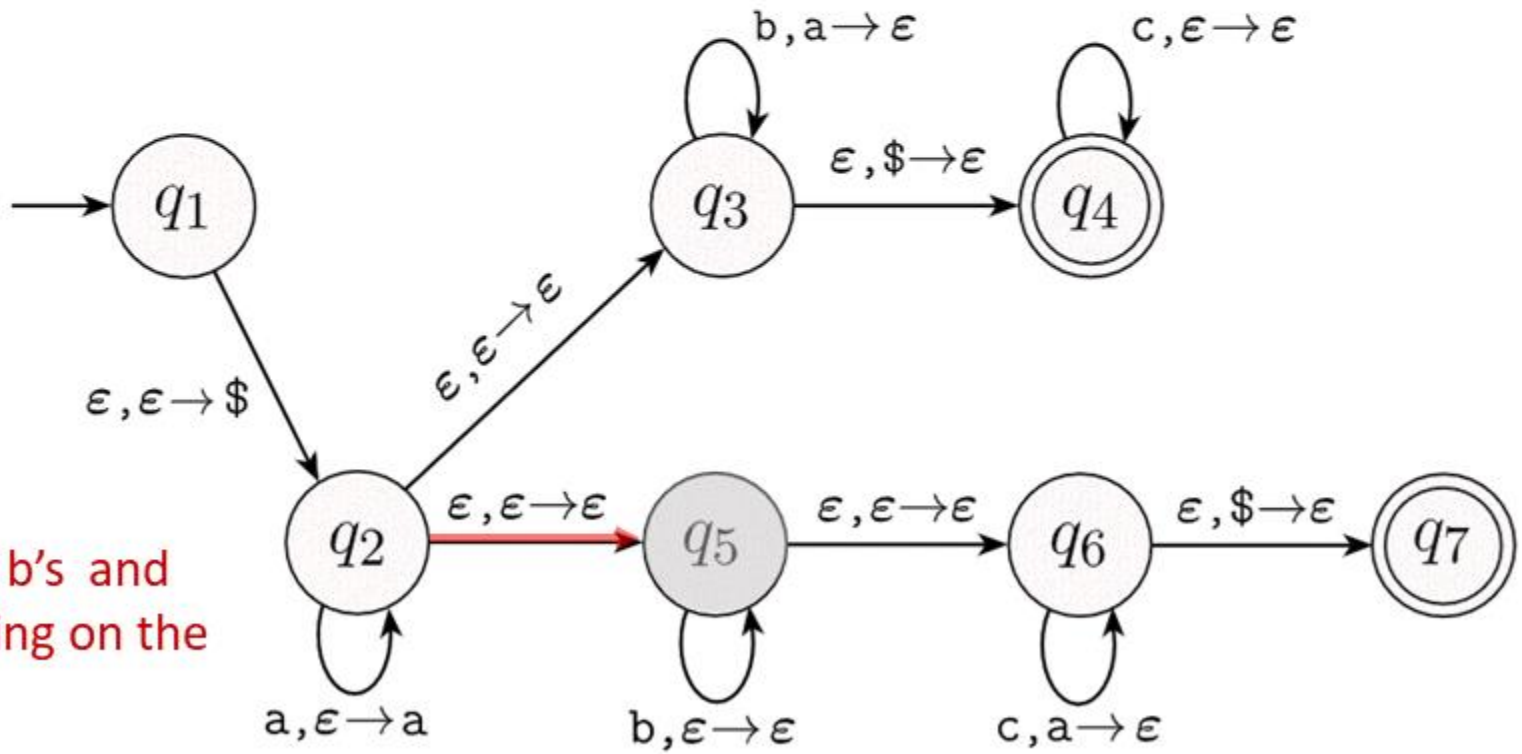


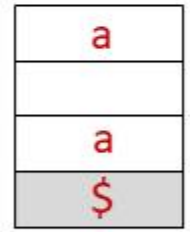
- Reading a's and pushing the same amount of a's onto the stack.

EXAMPLES OF PUSHDOWN AUTOMATA

- **Example 2:** Find the PDA that recognizes the language

$$\{\mathrm{a}^i\mathrm{b}^j\mathrm{c}^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}.$$



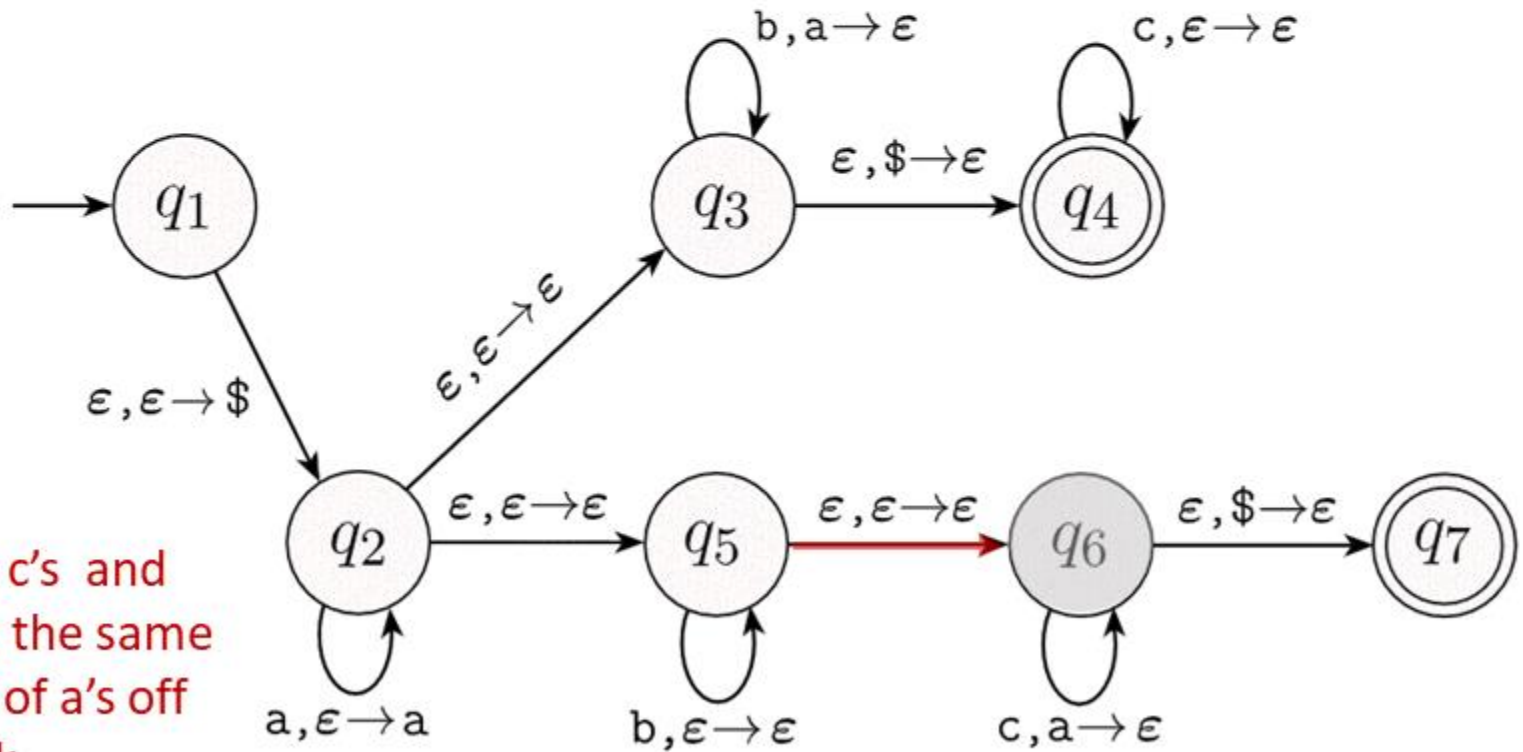


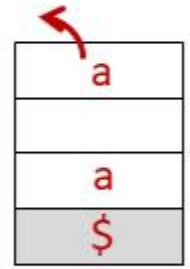
- Reading b's and do nothing on the stack.

EXAMPLES OF PUSHDOWN AUTOMATA

- **Example 2:** Find the PDA that recognizes the language

$$\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}.$$



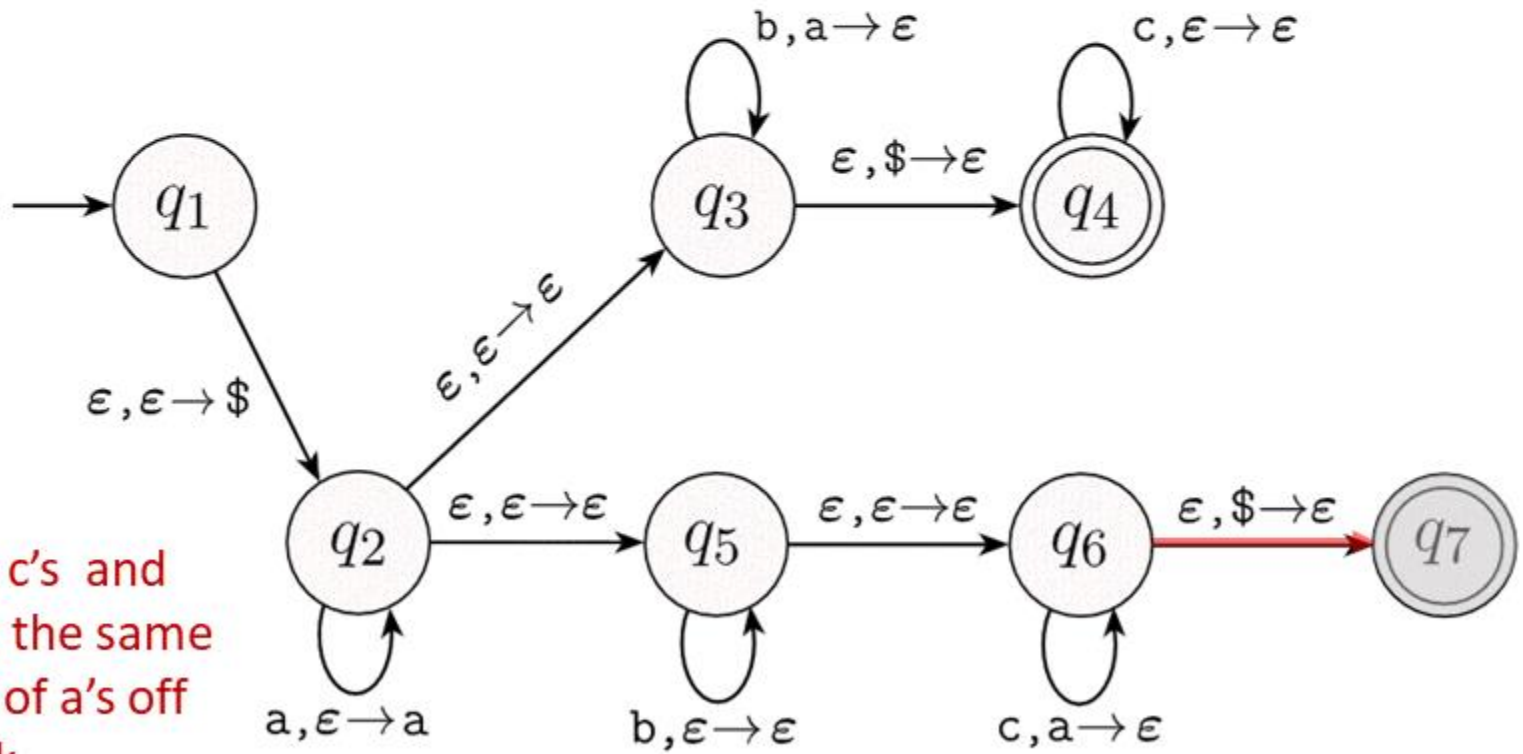


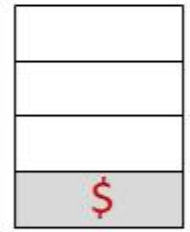
- Reading c's and popping the same amount of a's off the stack.

EXAMPLES OF PUSHDOWN AUTOMATA

- **Example 2:** Find the PDA that recognizes the language

$$\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}.$$





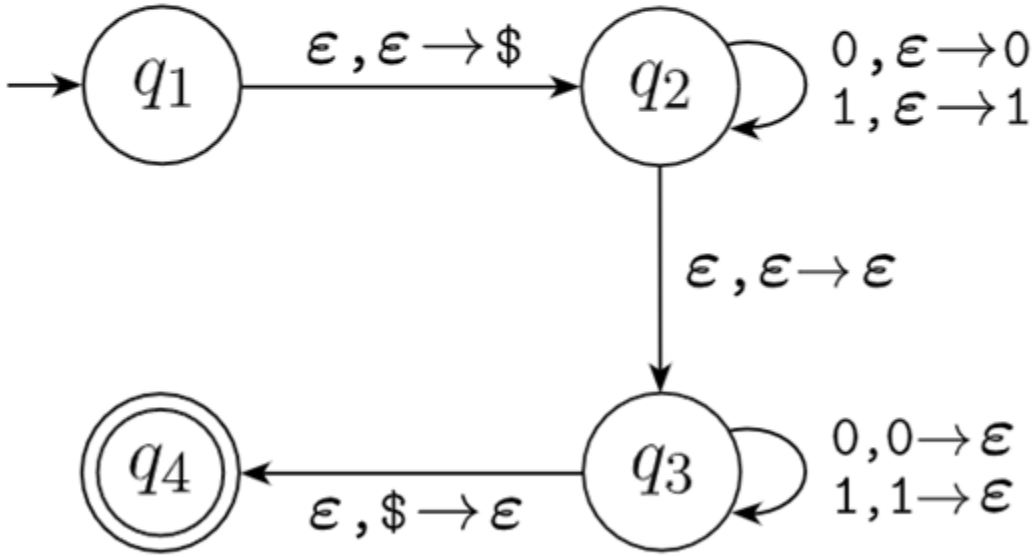
- Reading c's and popping the same amount of a's off the stack.

EXAMPLES OF PUSHDOWN AUTOMATA

- **Example 3:** Find the PDA that recognizes the language

$$\{ww^{\mathcal{R}} | w \in \{0,1\}^*\}$$

NOLEMON | NOMELON



EQUIVALENCE OF PDA WITH CFG

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

- Part1: If a language is context free, then some pushdown automaton recognizes it.

- Part2: If a pushdown automaton recognizes some language, then it is context free.

EQUIVALENCE OF PDA WITH CFG

- Part1: If a language is context free, then some pushdown automaton recognizes it.

- PROOF IDEA:
- Let A be a CFL, Then A has a CFG **G**, generating it.
- Using the following example we show how to convert **G** into an equivalent PDA, **P**.

- **Example:** Grammar **G**

$$S \rightarrow BS|A \quad : R1, R2$$
$$A \rightarrow 2A|\varepsilon \quad : R3, R4$$
$$B \rightarrow BB1|0 \quad : R5, R6$$

Using **left-most derivation** for generating **001** will be :

$$S \Rightarrow BS \Rightarrow BB1S \Rightarrow 0B1S \Rightarrow 001S \Rightarrow 001A \Rightarrow 001$$

R1 R5 R6 R6 R2 R4

EQUIVALENCE OF PDA WITH CFG

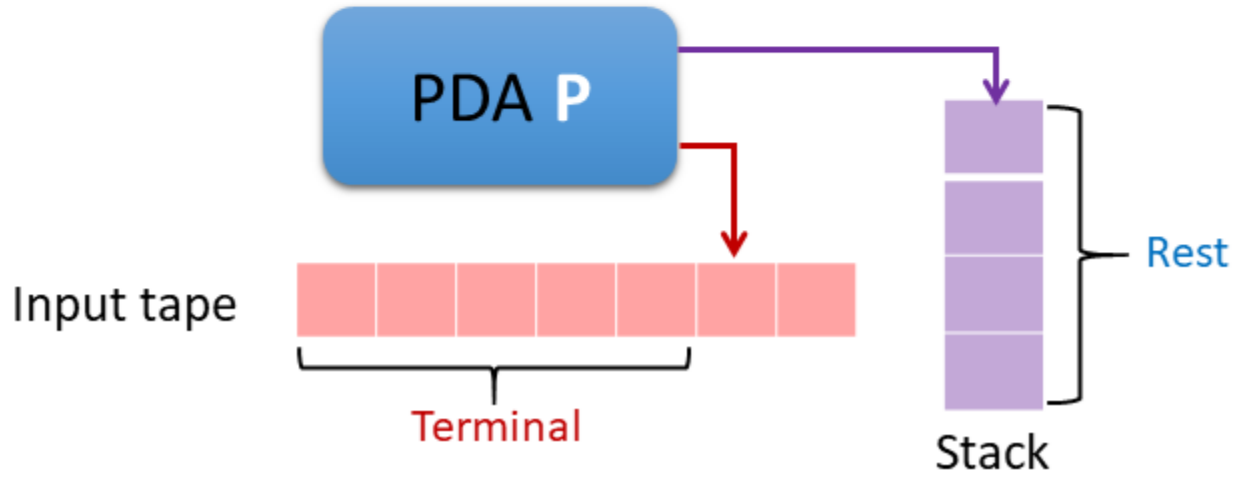
- PROOF IDEA: (cont.)

- In general form all of intermediate string in the derivation for generating a string will be in form

$$0001100101\underbrace{01}_{\text{Terminal}}\underbrace{SA01SSA00SSAA0}_{\text{Rest}}$$

(contains terminals and nonterminals)

- The **PDA** P will work by accepting its input w, if G generates that input,
  - by determining whether there is a derivation for w. (left-most derv.)

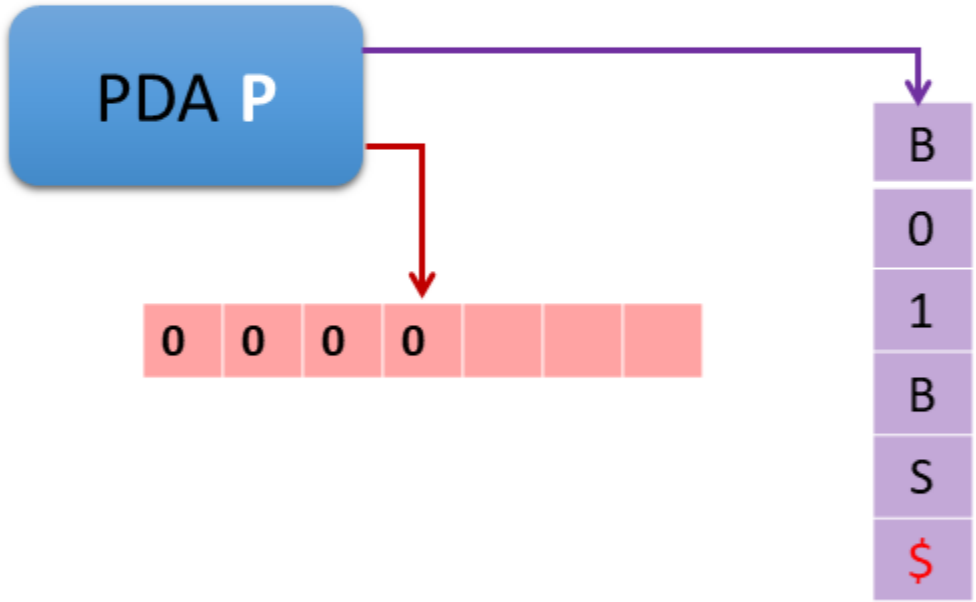


EQUIVALENCE OF PDA WITH CFG

- PROOF IDEA: (cont.)

- Let's see how we can store a an **intermediate string** (sentential form) in the derivation in a pushdown automaton.

$$S \overset{*}{\Rightarrow} 0000B01BS \Rightarrow \cdots$$



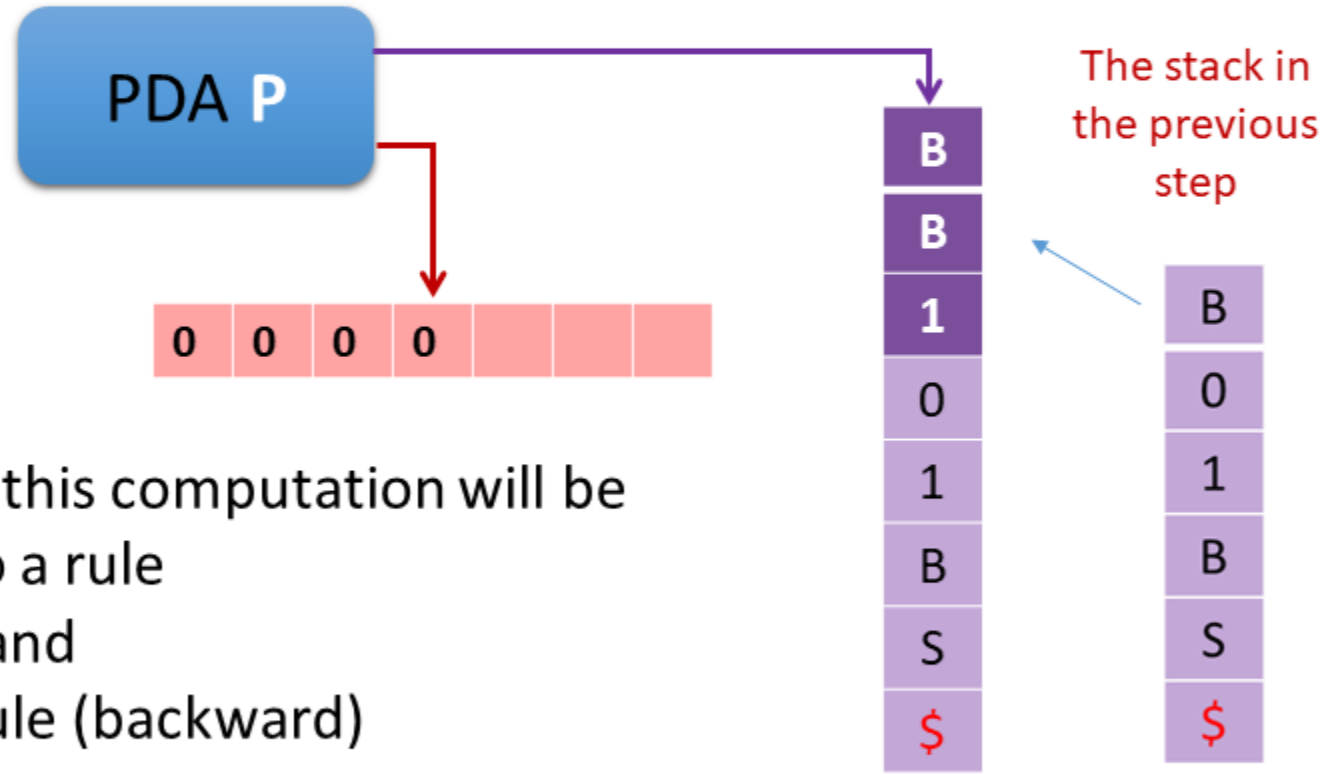
EQUIVALENCE OF PDA WITH CFG

- PROOF IDEA: (cont.)

- At each step of the derivation, we expand the leftmost nonterminal using the rules.

$$S \overset{*}{\Rightarrow} 0000B01BS \Rightarrow \cdots$$

Rule: $\mathrm{B} \longrightarrow \mathrm{BB1}$



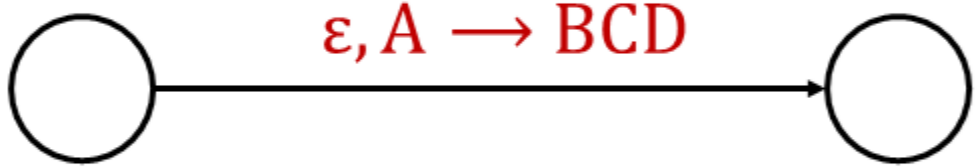
- so the steps that we follow in this computation will be
  - We match the stack top to a rule
  - Pop the top of the stacks and
  - push the left side of the rule (backward)

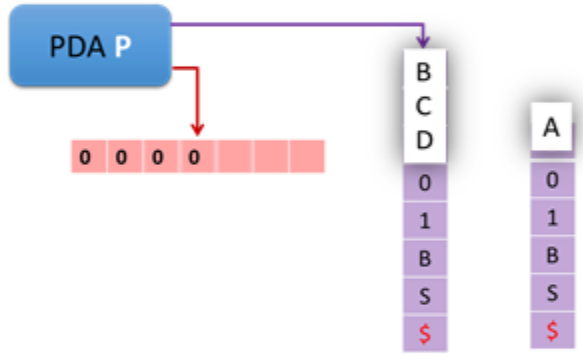
- Since PDA is nondeterministic, all rules at the same time will be examined.

EQUIVALENCE OF PDA WITH CFG

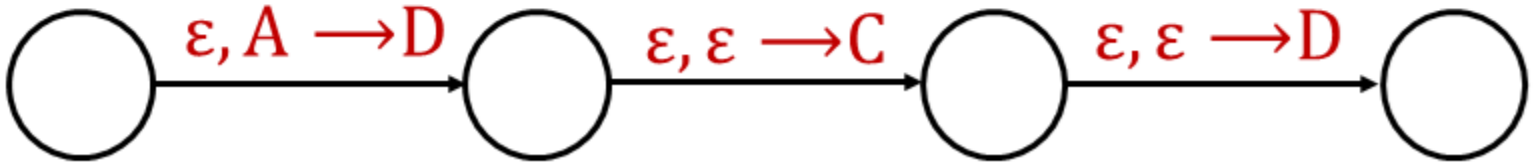
- PROOF IDEA: (cont.)

- How can we add the following rule in a PDA?
- **Rule**: $A \longrightarrow BCD$ where A,B,C, and D are nonterminals





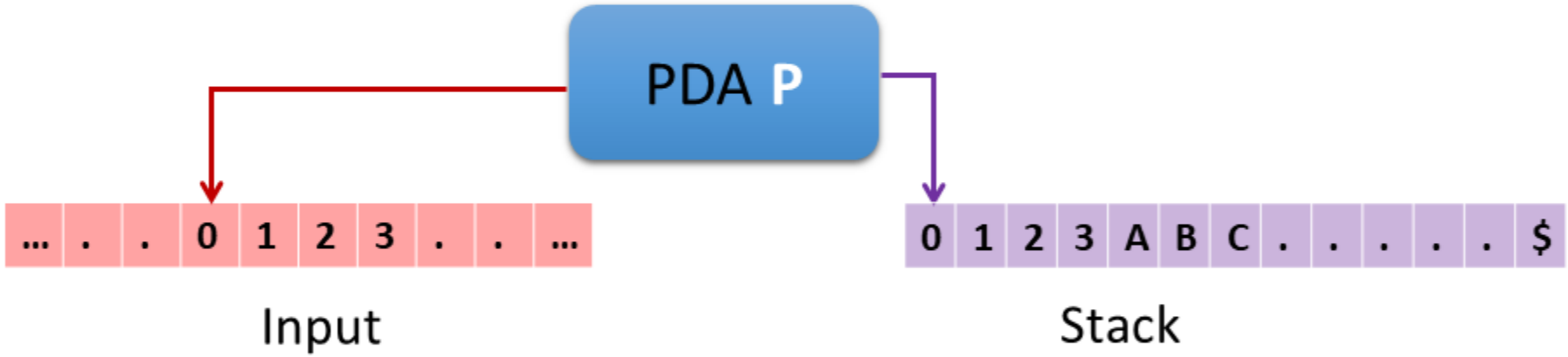
- We are not allowed to push multiple symbols.

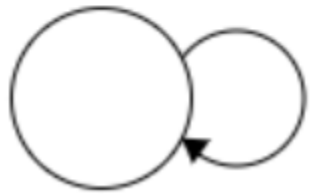


EQUIVALENCE OF PDA WITH CFG

- PROOF IDEA: (cont.)

- What will be the **transition** if there is a **terminal** on top of the stack?





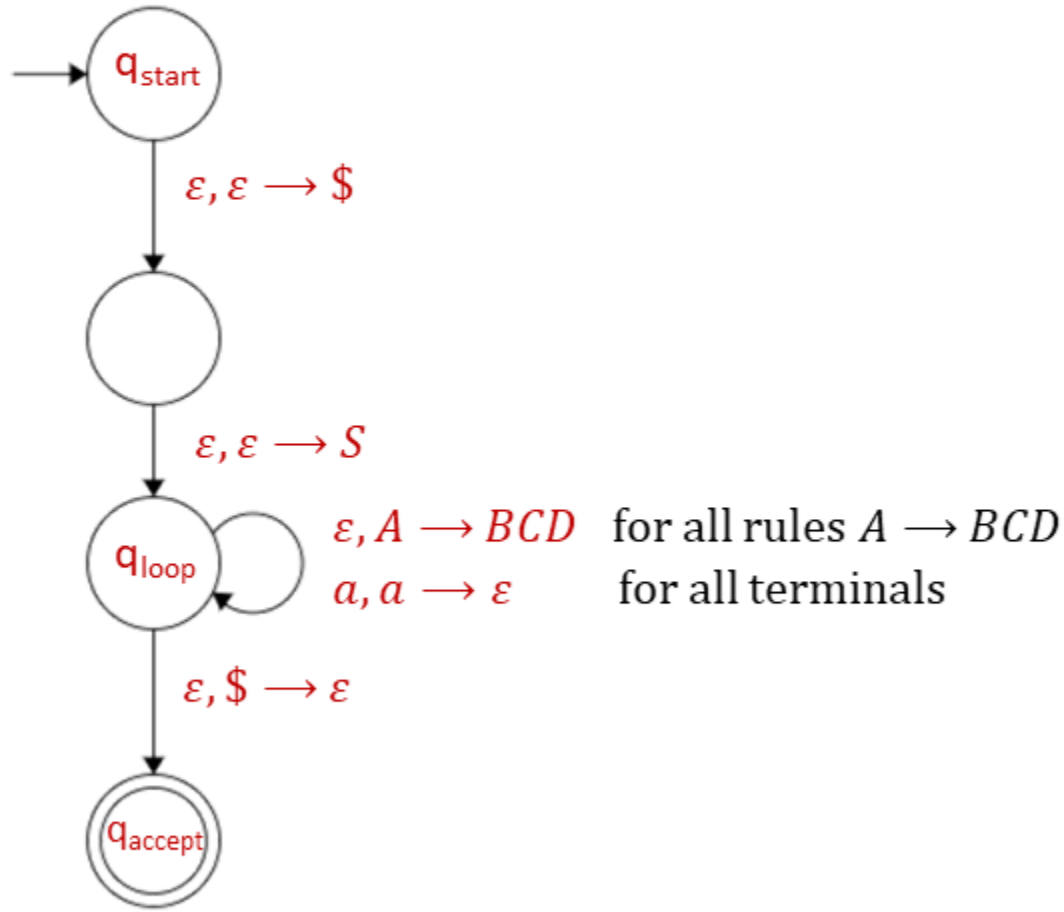
$0,0 \longrightarrow \varepsilon$
$1,1 \longrightarrow \varepsilon$
$2,2 \longrightarrow \varepsilon$
$3,3 \longrightarrow \varepsilon$

For all symbols in alphabet.
(nonterminals)

EQUIVALENCE OF PDA WITH CFG

- PROOF IDEA: (cont.)

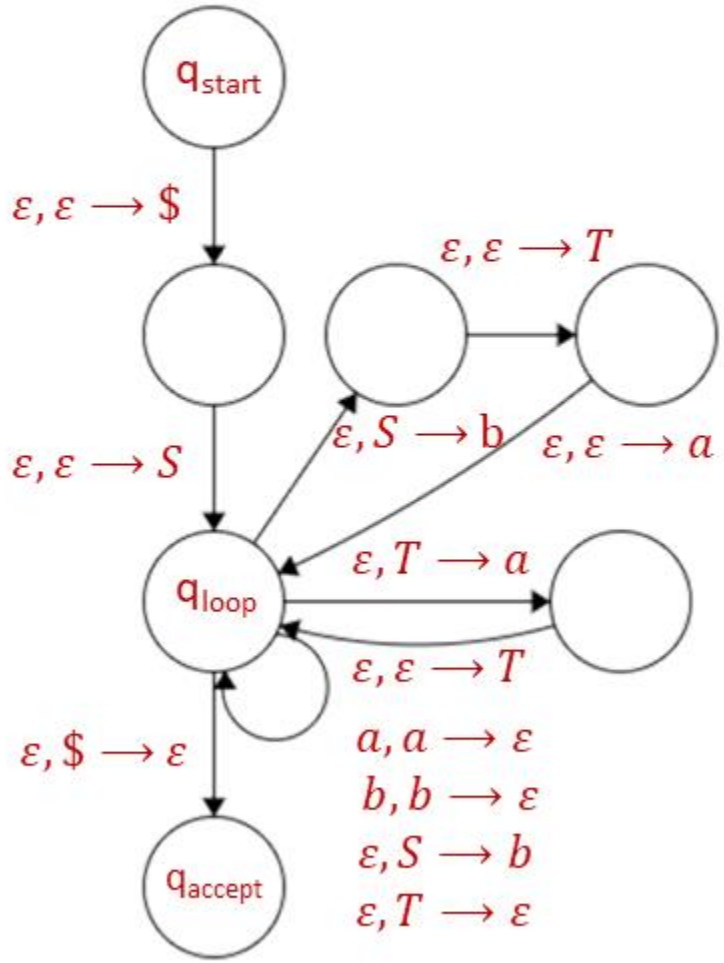
- The Final PDA



EQUIVALENCE OF PDA WITH CFG

- Example: construct aPDA P1 from the following CFG G.

$S \rightarrow aTb|b$
$T \rightarrow Ta|\varepsilon$



EQUIVALENCE OF PDA WITH CFG

- Part2: If a pushdown automaton recognizes some language, then it is context free.

EQUIVALENCE OF PDA WITH CFG

- Let's see one example that shows how we can convert an NFA to a CFG.
  - (**Note:** Any Regular language has CFG.)

EQUIVALENCE OF PDA WITH CFG

- Let's see one example that shows how we can convert an NFA to a CFG.
  - (**Note:** Any Regular language has CFG.)

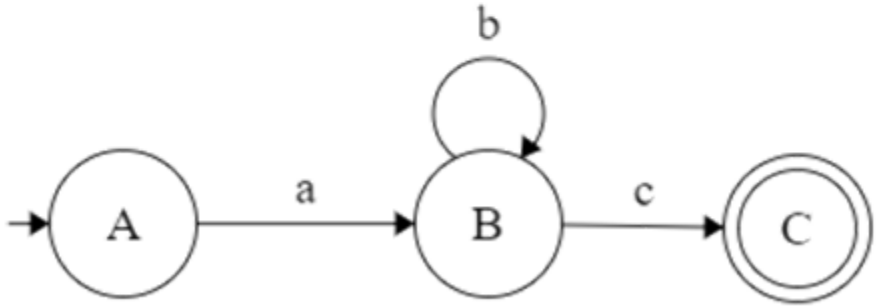
- Follow the following steps:
  - Make a variable for each state.
  - Make the variable for the starting state the **starting variable**.
  - Make a rule for each edge.
  - Add an ε-rule for each accept state.

EQUIVALENCE OF PDA WITH CFG

- Let's see one example that shows how we can convert an NFA to a CFG.
  - (**Note:** Any Regular language has CFG.)

- Follow the following steps:
  - Make a variable for each state.
  - Make the variable for the starting state the **starting variable**.
  - Make a rule for each edge.
  - Add an ε-rule for each accept state.

- **Example:** Find a CFG for the following Machine.



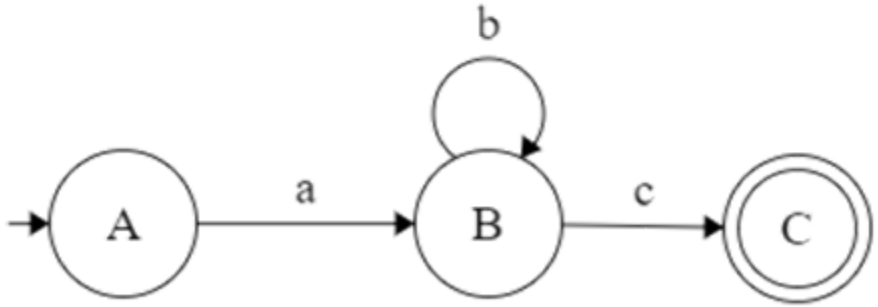
NFA M1

EQUIVALENCE OF PDA WITH CFG

- Let's see one example that shows how we can convert an NFA to a CFG.
  - (**Note:** Any Regular language has CFG.)

- Follow the following steps:
  - Make a variable for each state.
  - Make the variable for the starting state the **starting variable**.
  - Make a rule for each edge.
  - Add an ε-rule for each accept state.

- **Example:** Find a CFG for the following Machine.



NFA M1

- $A \rightarrow aB$
- $B \rightarrow bB \mid cC$
- $C \rightarrow \varepsilon$

CFG G1

$$A \rightarrow aB \rightarrow abB \rightarrow abcC \rightarrow abc\varepsilon = abc$$

EQUIVALENCE OF PDA WITH CFG

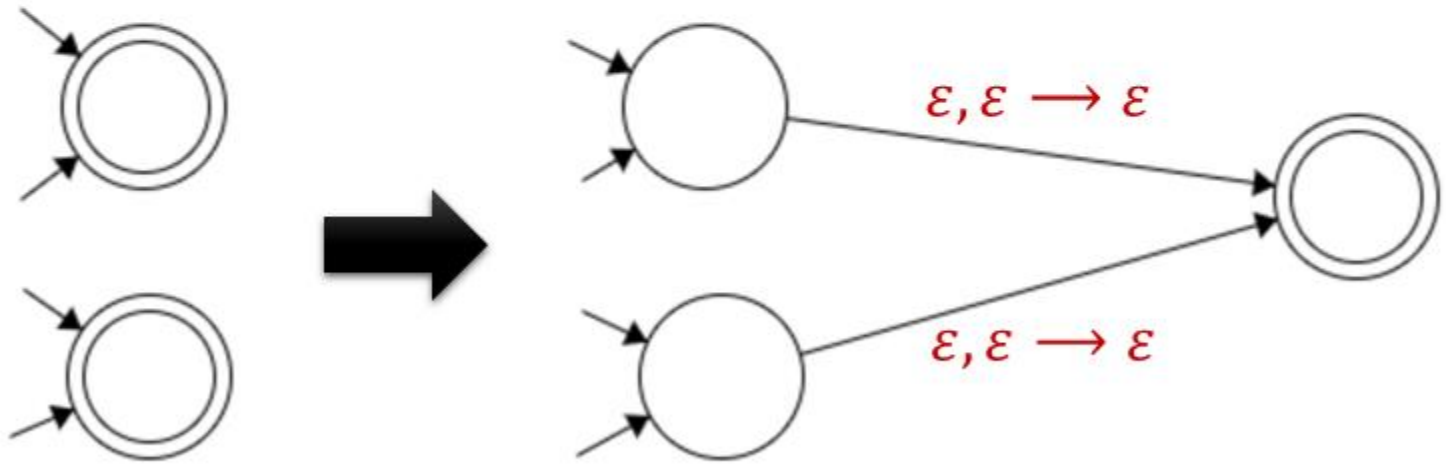
- Part2: If a pushdown automaton recognizes some language, then it is context free.

- Proof idea:
- We have a PDA P, and we want to make a CFG G that generates all the strings that P accepts.
- G should **generate a string** if that string causes the PDA to go from its start state to an accept state.

- **Steps:**
  - **Step 1:** simplify the PDA
  - **Step2:** Building CFG

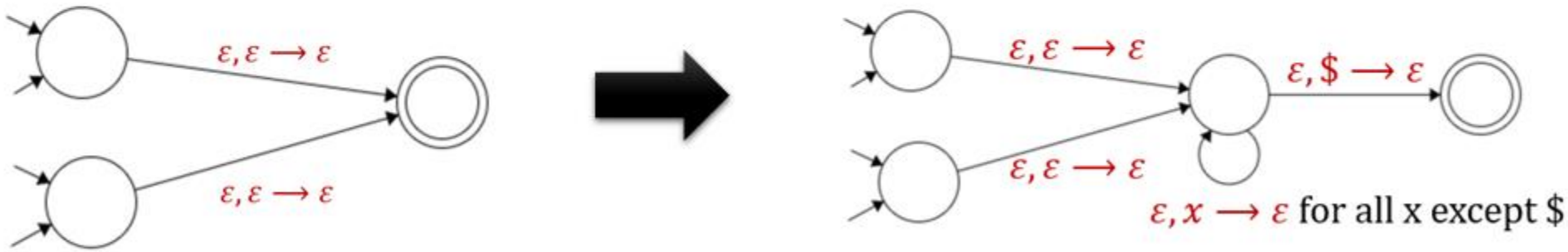
EQUIVALENCE OF PDA WITH CFG

- Proof idea: (cont.)
- **Step1:** Simplification by modifying P as below to give it the following three features:
  1. It has a single accept state, q_{accept}.



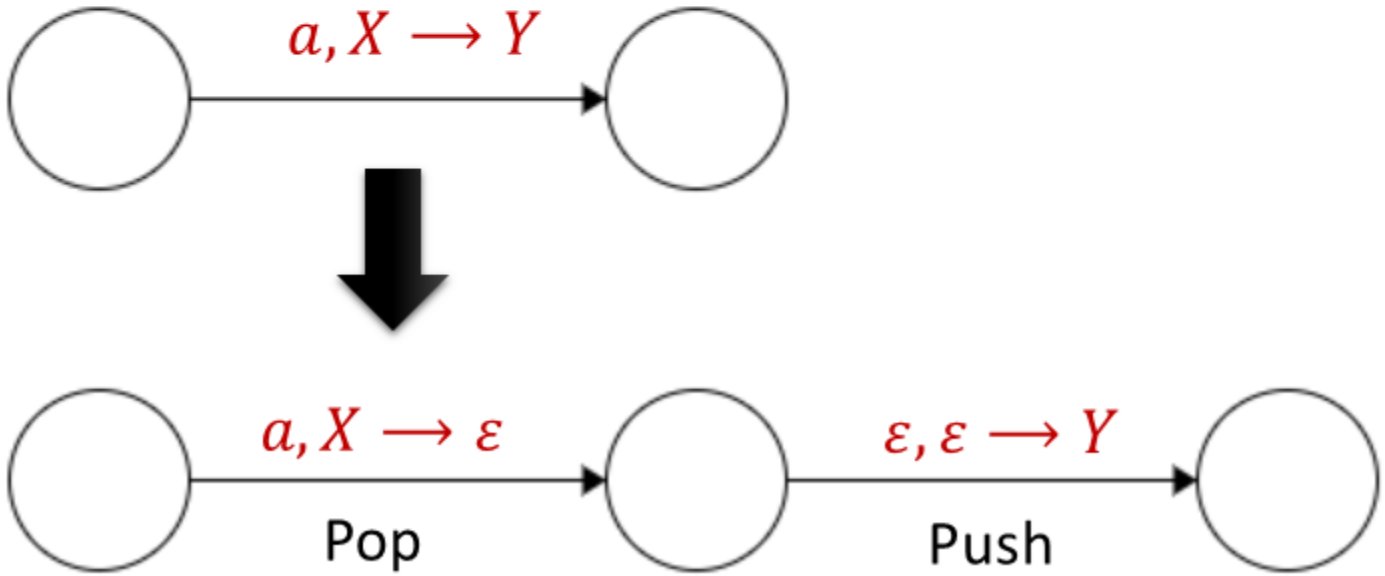
EQUIVALENCE OF PDA WITH CFG

- Proof idea: (cont.)
- **Step1:** Simplification by modifying P as below to give it the following three features:
  2. It empties its stack before accepting.



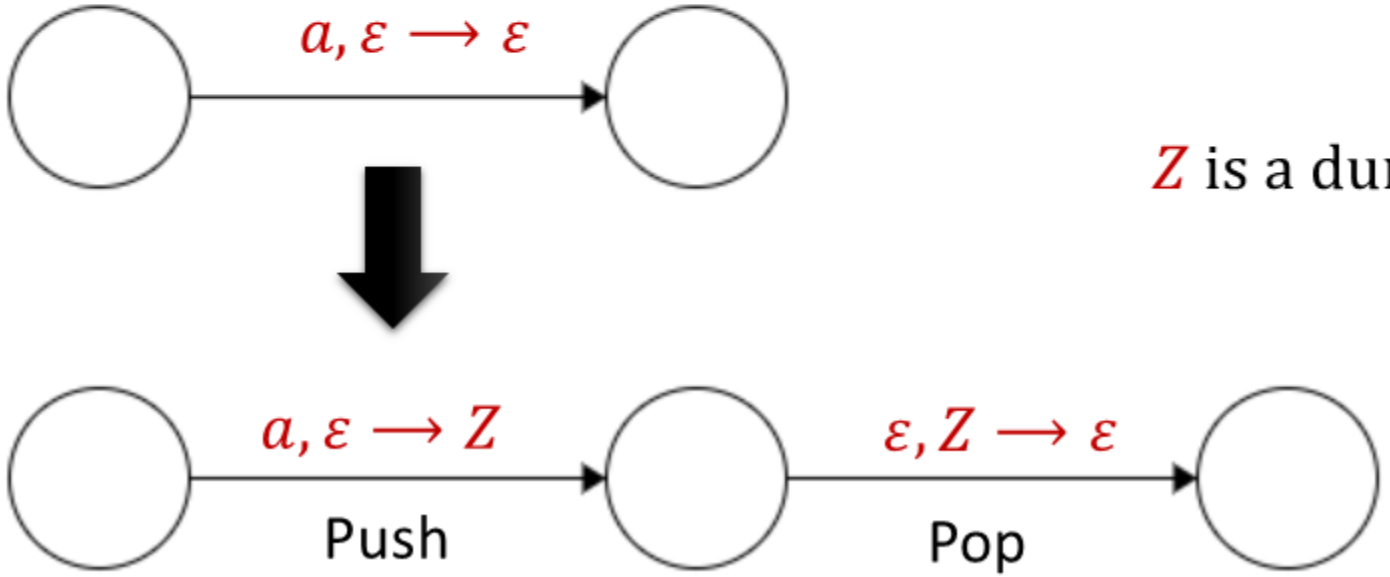
EQUIVALENCE OF PDA WITH CFG

- Proof idea: (cont.)
- **Step1:** Simplification by modifying P as below to give it the following three features:
  3. Each transition either pushes a symbol onto the stack (a push move) or pops one off the stack (a pop move), but it does not do both at the same time.



EQUIVALENCE OF PDA WITH CFG

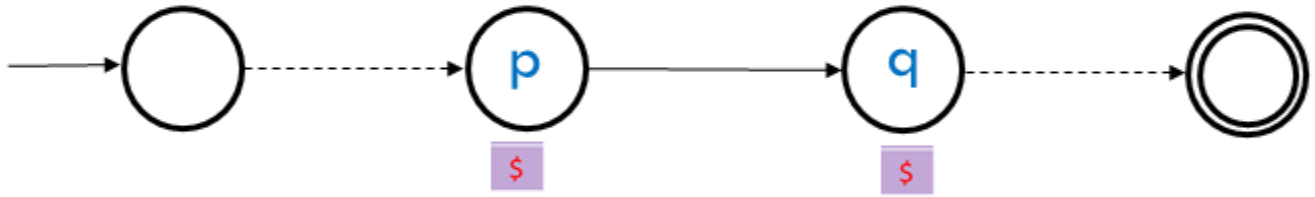
- Proof idea: (cont.)
- **Step1:** Simplification by modifying P as below to give it the following three features:
  3. Each transition either pushes a symbol onto the stack (a push move) or pops one off the stack (a pop move), but it does not do both at the same time.



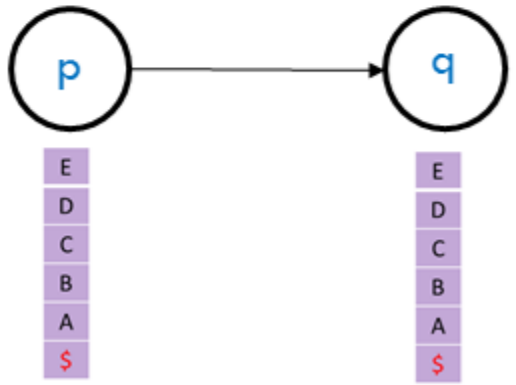
Z is a dummy symbol.

EQUIVALENCE OF PDA WITH CFG

- Proof idea: (Cont.)
- **Step2: Building CFG**
- For **each pair of states p and q** in P, the grammar will have a **variable** A_{pq}.
  - This variable **generates all the strings** that can take P from p with an empty stack to q with an empty stack.



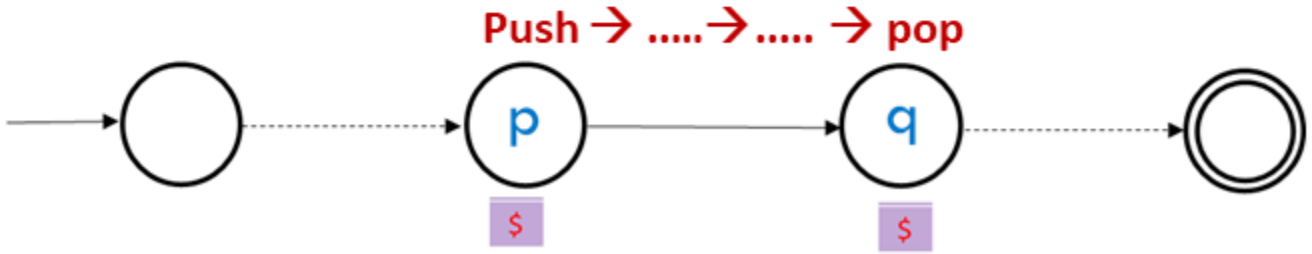
  - such strings can also take P from p to q, regardless of the stack contents at p, leaving the stack at q in the same condition as it was at p.



EQUIVALENCE OF PDA WITH CFG

- Proof idea: (Cont.)
- **Step2:**

- To design G, we must understand **how P operates** on the strings generated by A_{pq}.
  - For any such string x, P's **first move** on x must be a **push**,
    - Since **stack is empty**.
  - Similarly, the **last move** on x must be a **pop**.
    - Since the stack should be empty.

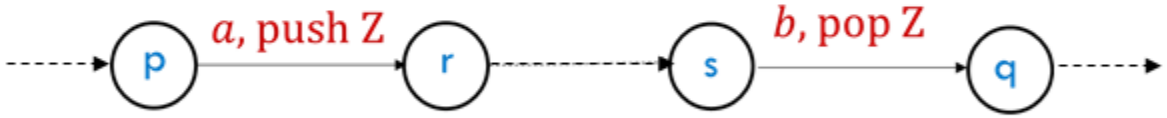


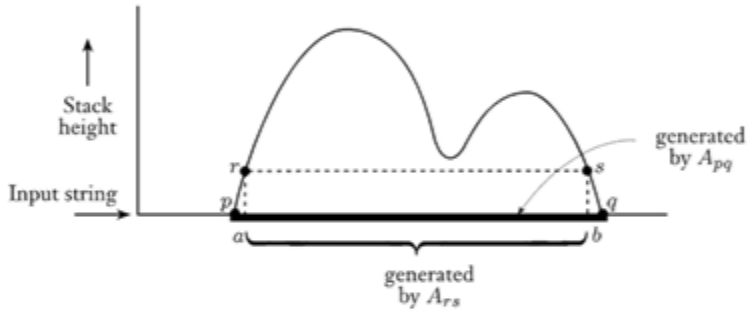
EQUIVALENCE OF PDA WITH CFG

- Proof idea: (Cont.)
- **Step2:**

- Two possibilities occur during P's computation on x.

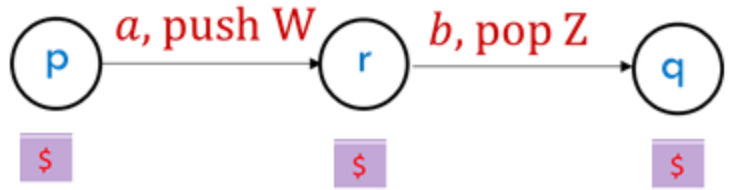
1. the **symbol popped at the end is** the **symbol that was pushed at the beginning**.

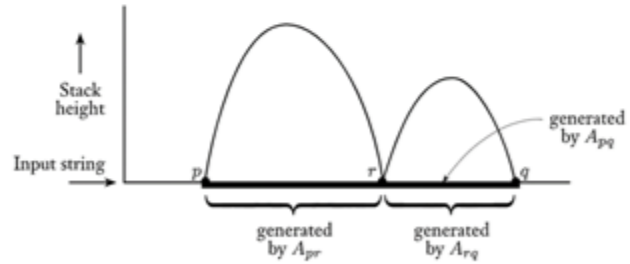




$A_{pq} \longrightarrow \mathrm{a}\ A_{\mathrm{rs}}\ \mathrm{b}$

2. the **symbol popped at the end is not** the **symbol that was pushed at the beginning**.





$A_{pq} \longrightarrow A_{\mathrm{pr}}\ A_{\mathrm{rq}}$

EQUIVALENCE OF PDA WITH CFG

- Part2: If a pushdown automaton recognizes some language, then it is context free.

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

1. For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, if $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε), put the rule $A_{pq} \to aA_{rs}b$ in G.
2. For each $p, q, r \in Q$, put the rule $A_{pq} \to A_{pr}A_{rq}$ in G.
3. Finally, for each $p \in Q$, put the rule $A_{pp} \to \varepsilon$ in G.

Chapter 2

2.2 Non-Context-Free Grammars

Introduction

- We have shown that language $A = \{a^n b^n | n \geq 0\}$ is a CFL.
- How about $B = \{a^n b^n c^n | n \geq 0\}$?

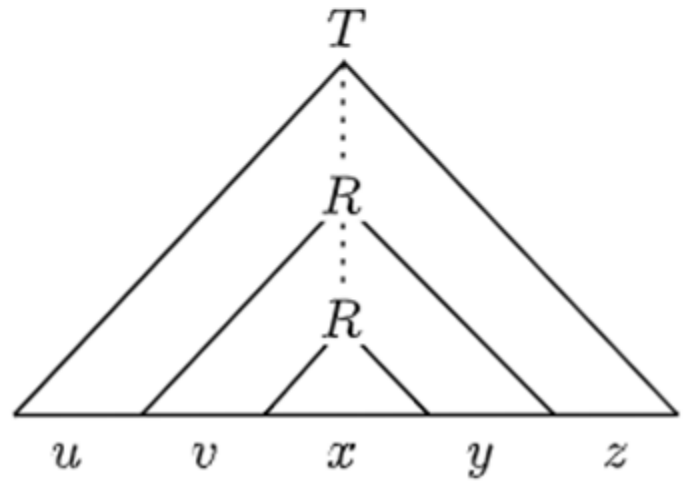
In this section:

- A similar pumping lemma for context-free languages is presented.
  - It states that **every context-free language** has a special value called the pumping length such that **all longer strings** in the language can be "pumped."

Pumping Lemma for CFL

Theorem: **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 piece** $s = uvxyz$ satisfying the conditions

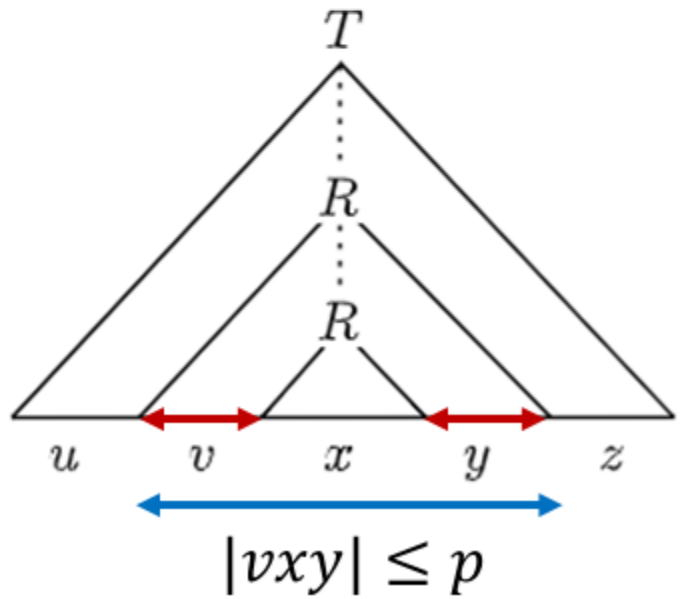


Pumping Lemma for CFL

Theorem: **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 piece** $s = uvxyz$ satisfying the conditions

1. for each $i \geq 0, uv^i xy^i z \in A$,
2. $|vy| > 0$, and → either v or y is not the empty string.
3. $|vxy| \leq p$. → the pieces v, x, and y together have length at most p.



Theorem: Pumping Lemma for CFL

Proof idea:

- Let A be a CFL and let G be a CFG that generates it. A= L(G)
- Let s be a **very long** string in A.

Theorem: Pumping Lemma for CFL

Proof idea:

- Let A be a CFL and let G be a CFG that generates it. A= L(G)
- Let s be a **very long** string in A.

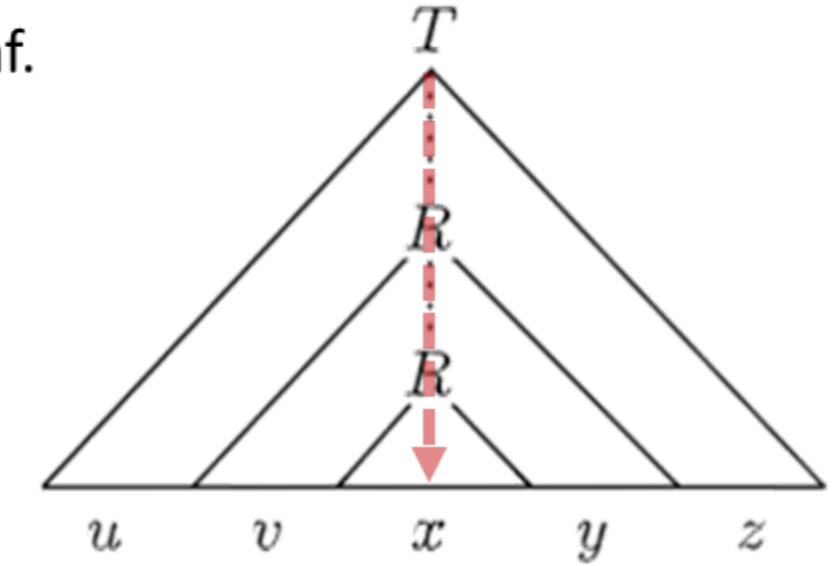
- Because s is in A,
  - it is derivable from G and so has a parse tree.
- The parse tree for s must be very tall
  - because s is very long.

Theorem: Pumping Lemma for CFL

Proof idea:

- Let A be a CFL and let G be a CFG that generates it. A= L(G)
- Let s be a **very long** string in A.

- Because s is in A,
  - it is derivable from G and so has a parse tree.
- The parse tree for s must be very tall
  - because s is very long.
- The parse tree must contain some long path from the start variable at the root of the tree to one of the terminal symbols at a leaf.



Theorem: Pumping Lemma for CFL

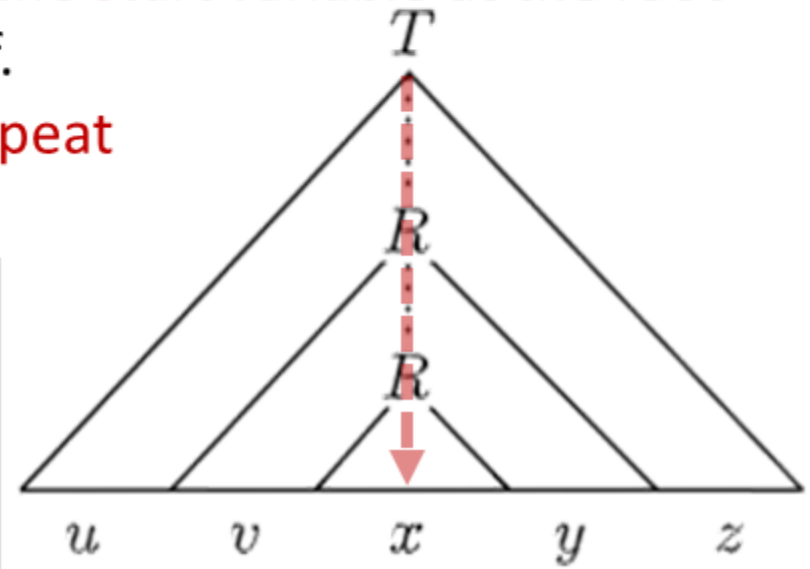
Proof idea:

- Let A be a CFL and let G be a CFG that generates it. A= L(G)
- Let s be a **very long** string in A.

- Because s is in A,
  - it is derivable from G and so has a parse tree.
- The parse tree for s must be very tall
  - because s is very long.
- The parse tree must contain some long path from the start variable at the root of the tree to one of the terminal symbols at a leaf.
- On this **long path**, some variable symbol R must repeat
  - because of the pigeonhole principle.

$T \to R \quad R \to aR\text{b}|\text{B} \quad B \to c$

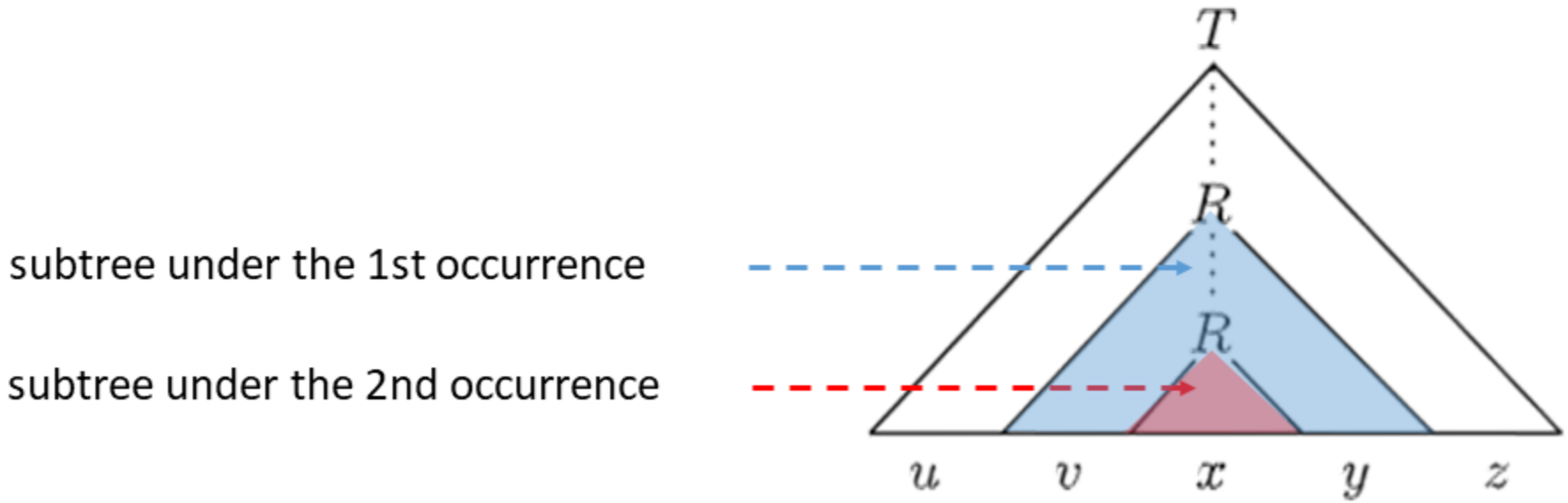
$T \to R \to aR\text{b} \to aB\text{b} \to ac\text{b}$



Theorem: Pumping Lemma for CFL

Proof idea: (cont.)

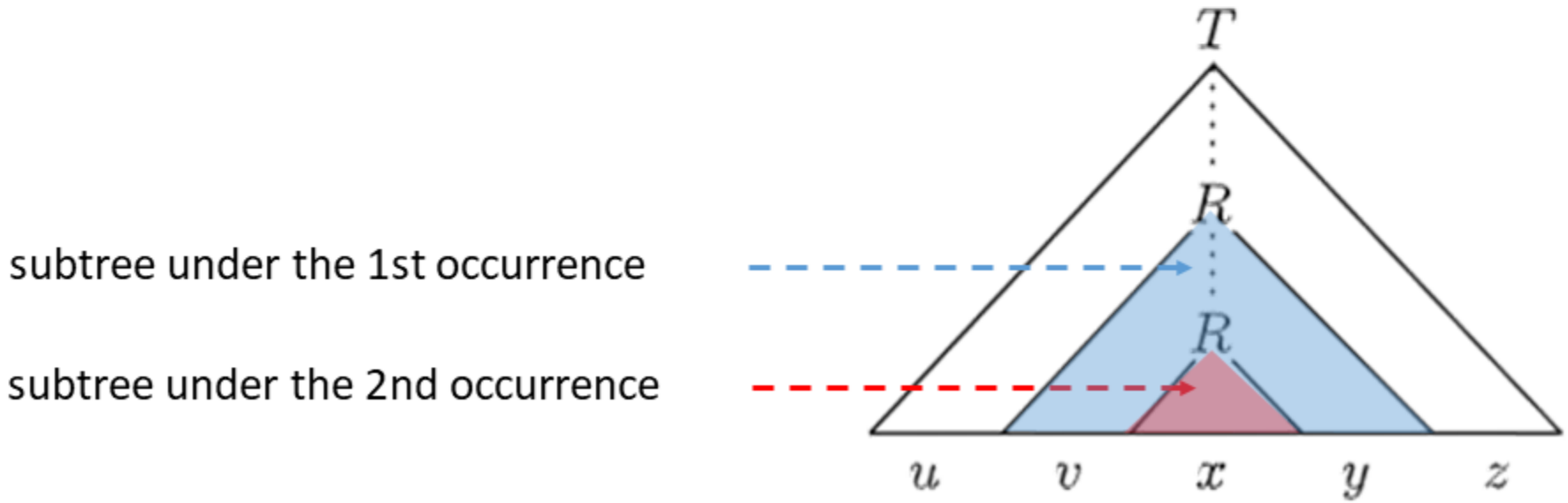
- This repetition allows us to replace the subtree under the second occurrence of R with the subtree under the first occurrence of R and still get a legal parse tree.



Theorem: Pumping Lemma for CFL

Proof idea: (cont.)

- This repetition allows us to replace the subtree under the second occurrence of R with the subtree under the first occurrence of R and still get a legal parse tree.
- Therefore, we may cut s into five pieces $uvxyz$ as the figure indicates.
- And we may repeat the second and fourth pieces
  - and obtain a string still in the language.

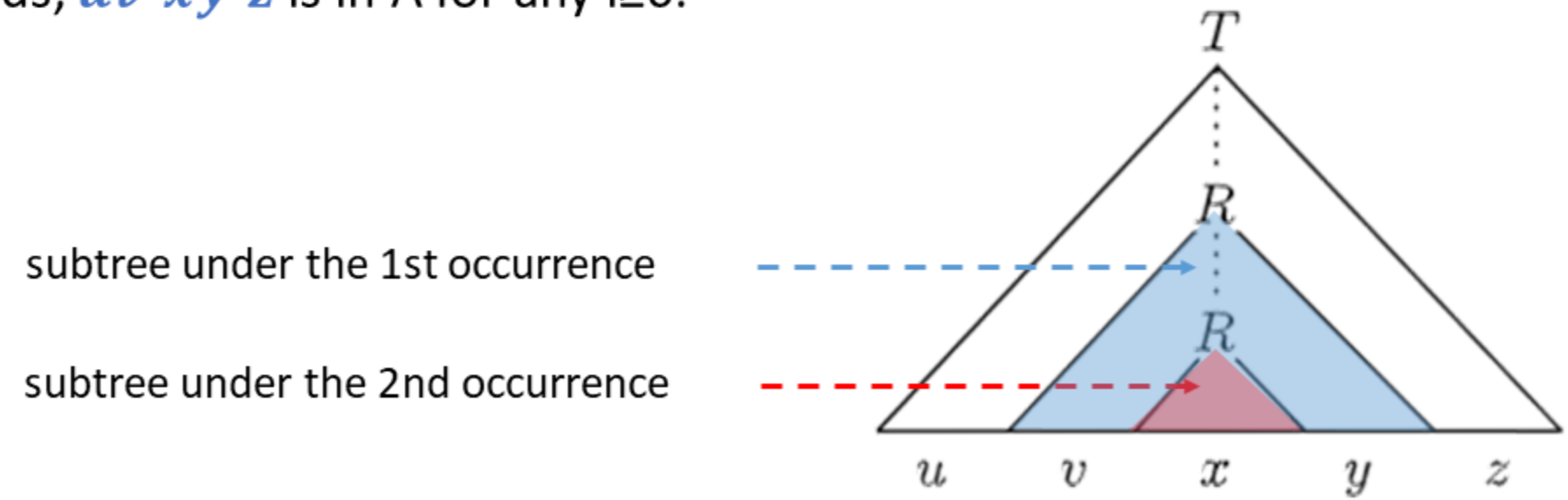


Theorem: Pumping Lemma for CFL

Proof idea: (cont.)

- This repetition allows us to replace the subtree under the second occurrence of R with the subtree under the first occurrence of R and still get a legal parse tree.
- Therefore, we may cut s into five pieces $uvxyz$ as the figure indicates.
- And we may repeat the second and fourth pieces
  - and obtain a string still in the language.

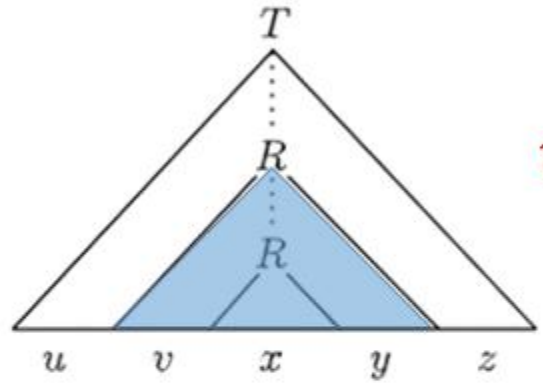
- In other words, uv^ixy^iz is in A for any i≥0.



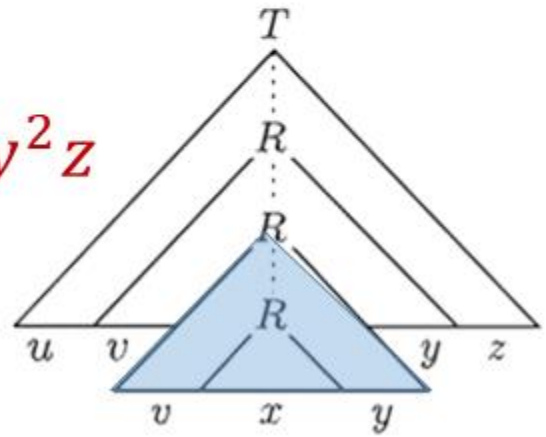
Introduction

Proof idea: (cont.)

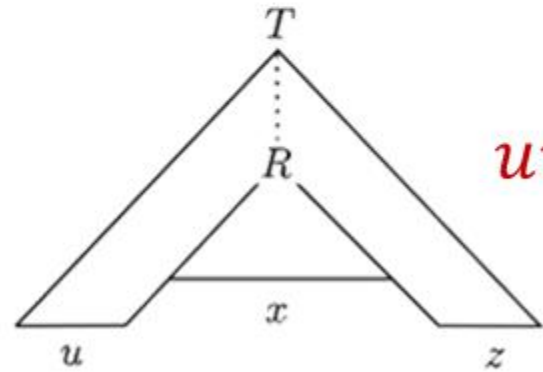
- In other words, $uv^i xy^i z$ is in A for any i≥0.



uv^1xy^1z



uv^2xy^2z



uv^0xy^0z

Example 1:

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

Example 1:

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

Proof by Contradiction:

- 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 $s = a^p b^p c^p$.

Example 1:

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

Proof by Contradiction:

- 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**.

Example 1:

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

Proof by Contradiction:

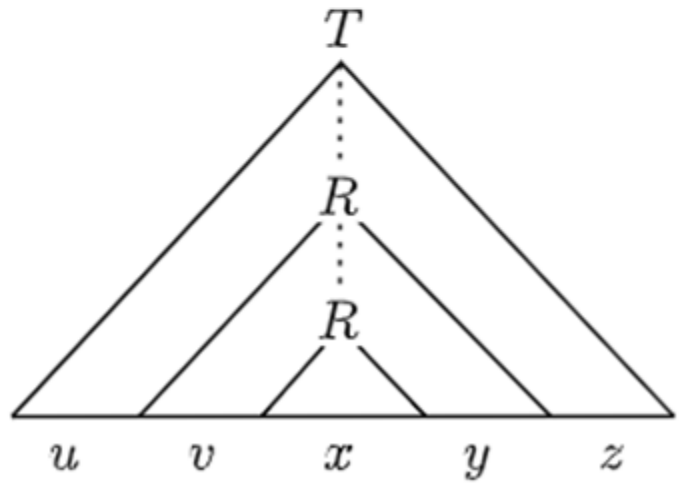
- We assume that B is a CFL and obtain a contradiction.
- Let p be the pumping length for B that is guaranteed to exist by the pumping lemma.
- Select the string $s = a^p b^p c^p$.
- Clearly s is a member of B and of length at least p.
- The **pumping lemma** states that s can be **pumped**,
  - but 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

Proof by Contradiction: (Cont.)

- First, **condition 2** stipulates that either v or y is nonempty. $(|vy| > 0)$
- Then we consider one of two cases:
  1. **When both v and y contain only one type of alphabet symbol**
  2. **When either v or y contains more than one type of symbol**

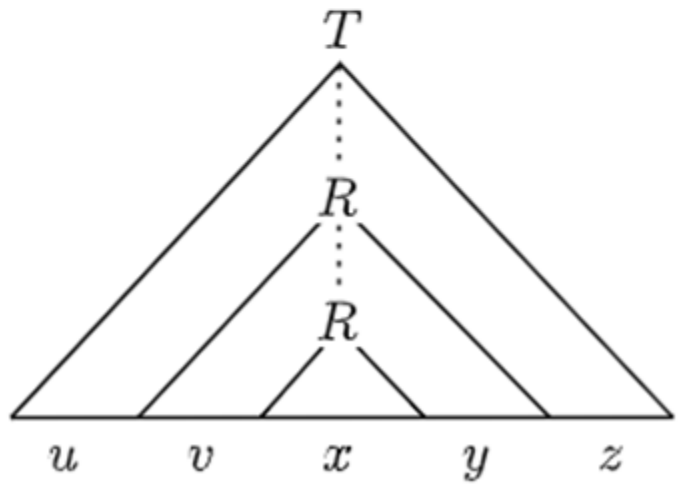
$$B = \{a^n b^n c^n | n \geq 0\}$$



Proof by Contradiction: (Cont.)

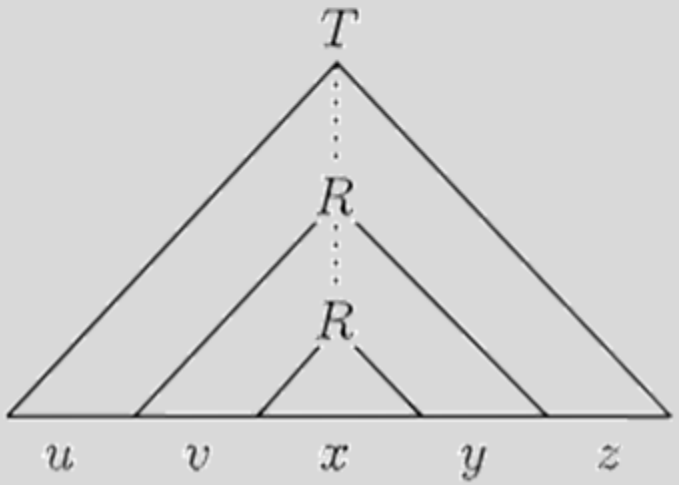
- First, **condition 2** stipulates that either v or y is nonempty. $(|vy| > 0)$
- Then we consider one of two cases:
  1. **When both v and y contain only one type of alphabet symbol,**
     - v does not contain both **a's** and **b's** or both **b's** and **c's**, and the same holds for y.

$$B = \{a^n b^n c^n | n \geq 0\}$$



Proof by Contradiction: (Cont.)

- First, **condition 2** stipulates that either v or y is nonempty. $(|vy| > 0)$
- Then we consider one of two cases:
  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.



$$B = \{a^n b^n c^n | n \geq 0\}$$

Proof by Contradiction: (Cont.)

- First, **condition 2** stipulates that either v or y is nonempty. $(|vy| > 0)$
- Then we consider one of two cases:
  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.

- Example: aaaaa bbbbb ccccc or aaaaa bbbbb ccccc
  v y v y=ε

Proof by Contradiction: (Cont.)

2. **When either v or y contains 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.

   - Example: aaaa**ab**bbb**b**ccccc → aaaa**abab**bbb**bb**ccccc
     v y
     uv^2xy^2z

Example 2:

- Use the pumping lemma to show that the language $C = \{a^i b^j c^k | 0 \leq i \leq j \leq k\}$ is not context free.

Example 2:

- Use the pumping lemma to show that the language $C = \{a^i b^j c^k | 0 \leq i \leq j \leq k\}$ is not context free.

Proof by Contradiction:

- We assume that C is a CFL and obtain a contradiction.
- Let p be the pumping length for C that is guaranteed to exist by the pumping lemma.
- Select the string $s = a^p b^p c^p$.

Example 2:

- Use the pumping lemma to show that the language $C = \{a^i b^j c^k | 0 \leq i \leq j \leq k\}$ is not context free.

Proof by Contradiction:

- We assume that C is a CFL and obtain a contradiction.
- Let p be the pumping length for C 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 C 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

Proof by Contradiction: (Cont.)

- Let s = $uvxyz$.
- Then we consider two cases:
  1. **When both v and y contain only one type of alphabet symbol,**
  2. **When either v or y contains more than one type of symbol**

Proof by Contradiction: (Cont.)

- Let s = uvxyz.
- Then we consider two cases:
  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.
     - 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**.

Proof by Contradiction: (Cont.)

- Let s = uvxyz.
- Then we consider two cases:
  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.
       - **The a's do not appear.** Then we try **pumping down** to obtain the string $uv^0xy^0z = uxz$.
       - **Example:** aaaaa bbbbb ccccc → v=bb , y=cc

$$C = \{a^i b^j c^k | 0 \leq i \leq j \leq k\}$$

Proof by Contradiction: (Cont.)

- Let s = uvxyz.
- Then we consider two cases:
  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.
       - **The a's do not appear.** Then we try pumping down to obtain the string $uv^0xy^0z = uxz$.
       - **Example:** aaaaa bbbbb ccccc → v=bb , y=cc
       - That contain s 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.

Proof by Contradiction: (Cont.)

- Let s = uvxyz.
- Then we consider two cases:
  1. **When both v and y contain only one type of alphabet symbol,**
     - **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.
     - **Example:** aaaaa bbbbb ccccc → v=aa , y=cc

$$C = \{a^i b^j c^k | 0 \leq i \leq j \leq k\}$$

Proof by Contradiction: (Cont.)

- Let s = uvxyz.
- Then we consider two cases:
  1. **When both v and y contain only one type of alphabet symbol,**
     - **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.
     - **Example:** aaaaa bbbbb ccccc → v=aa , y=cc
     - 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.

Proof by Contradiction: (Cont.)

- Let s = uvxyz.
- Then we consider two cases:

1. **When both v and y contain only one type of alphabet symbol,**
   - **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.
   - **Example:** aaaaa bbbbb ccccc → v=aa , y=cc
   - If **a's appear**,
     - the string uv^2xy^2z contains more a's than b's,
     - so it is **not in C**.

$$C = \{a^i b^j c^k | 0 \leq i \leq j \leq k\}$$

Proof by Contradiction: (Cont.)

- Let s = uvxyz.
- Then we consider two cases:
  1. **When both v and y contain only one type of alphabet symbol,**
     - **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.

Proof by Contradiction: (Cont.)

- Let s = uvxyz.
- Then we consider two cases:
  2. **When either v or y contains more than one type of symbol**
     - Then the string uv^2xy^2z will not contain the symbols in the correct order.
       - so it is **not in C**, and a **contradiction** occurs.

Example 3:

- Use the pumping lemma to show that the language $D = \{ww | w \in \{0,1\}*\}$ is not context free.

Example 3:

- Use the pumping lemma to show that the language $D = \{ww | w \in \{0,1\}*\}$ is not context free.

Proof by Contradiction:

- We assume that D is a CFL and obtain a contradiction.
- Let p be the pumping length for D that is guaranteed to exist by the pumping lemma.
- Select the string $s = 0^p 1^p 0^p 1^p$.

Example 3:

- Use the pumping lemma to show that the language $D = \{ww | w \in \{0,1\}*\}$ is not context free.

Proof by Contradiction:

- We assume that D is a CFL and obtain a contradiction.
- Let p be the pumping length for D that is guaranteed to exist by the pumping lemma.
- Select the string $s = 0^p 1^p 0^p 1^p$.

Example 3:

- Use the pumping lemma to show that the language $D = \{ww | w \in \{0,1\}*\}$ is not context free.

Proof by Contradiction:

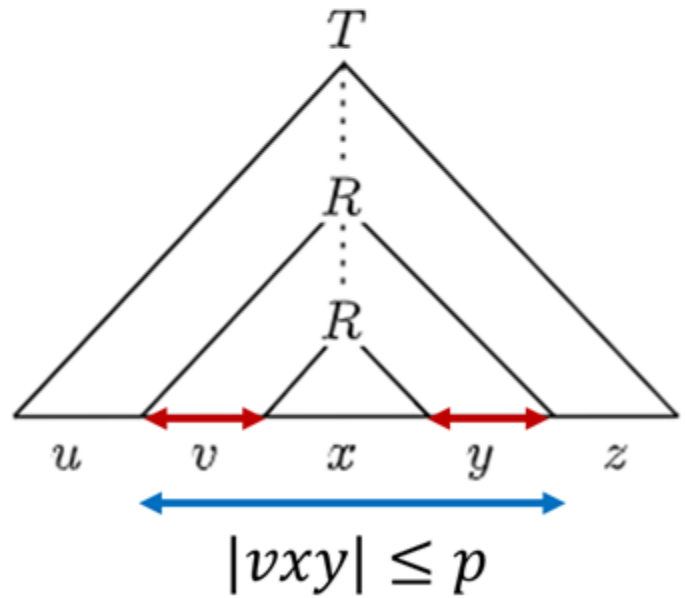
- We assume that D is a CFL and obtain a contradiction.
- Let p be the pumping length for D that is guaranteed to exist by the pumping lemma.
- Select the string $s = 0^p 1^p 0^p 1^p$.
- Clearly s is a member of D and of length at least p.
- The **pumping lemma** states that s can be **pumped**,
  - 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

Proof by Contradiction (Cont.):

- We use **condition 3** of the pumping lemma
  - It says that we can pump s by dividing $s = uvxyz$, where $|vxy| \leq p$.

- We show that if **condition 3** holds, then the **other conditions** must fail.



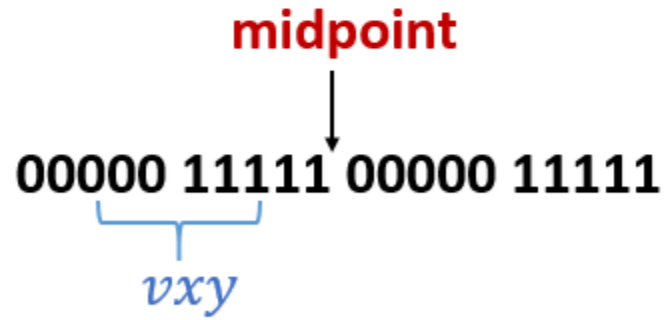
Proof by Contradiction (Cont.):

- We use **condition 3** of the pumping lemma
  - It says that we can pump s by dividing $s = uvxyz$, where $|vxy| \leq p$.

- We show that if condition 3 holds, then the other conditions must fail.

- **Case 1:** 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.

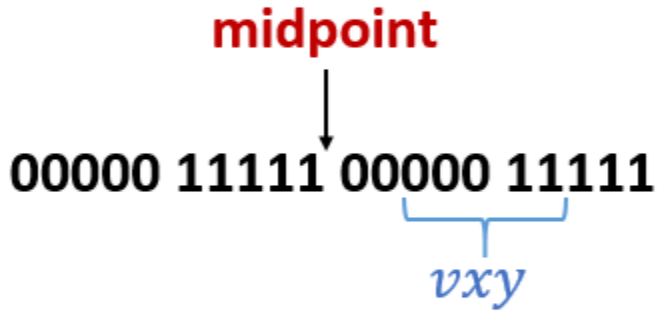
$$s = 0^p 1^p | 0^p 1^p$$



Proof by Contradiction (Cont.):

- **Case 2:** 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.

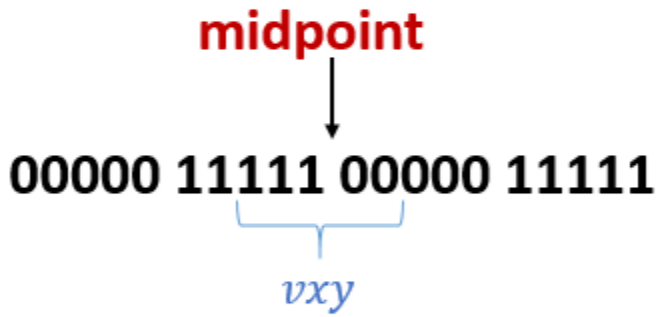
$$s = 0^p 1^p | 0^p 1^p$$



Proof by Contradiction (Cont.):

- **Case3:** 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.

$$s = 0^p1^p|0^p1^p$$



Then this string fails the conditions of pumping lemma.