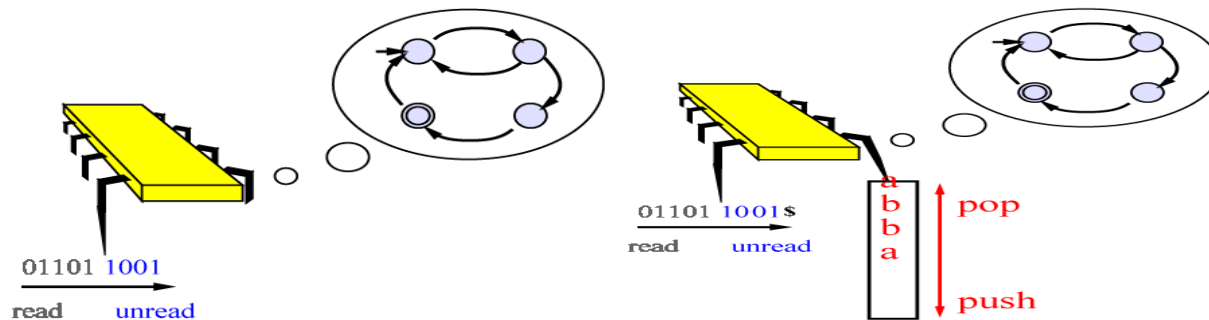


# Computational Models - Lecture 4

- **Context Free** Grammars  
and special case: Linear grammars
- **Pumping Lemma** for context free languages
- Non context free languages
- Push Down Automata (**PDA**)
- **Equivalence** of CFGs and PDAs (**statement only**)
- Chomsky Normal Form (maybe)



- Sipser's book, 2.1, 2.2 & 2.3

# Short Overview of the Course

So far we saw

- finite automata,
- regular languages,
- regular expressions,
- Myhill-Nerode theorem
- pumping lemma for regular languages.

We now introduce stronger machines and languages with more expressive power:

- pushdown automata,
- context-free languages,
- context-free grammars,
- pumping lemma for context-free languages.

# Formal Definitions

A context-free grammar is a 4-tuple  $(V, \Sigma, R, S)$  where

- $V$  is a finite set of variables,
- $\Sigma$  is a finite set of terminals,
- $R$  is a finite set of rules: each rule is a variable and a finite string of variables and terminals.
- $S$  is the start symbol.

# Formal Definitions

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

We write  $u \xRightarrow{*} v$  if  $u = v$  or

$$u \Rightarrow u_1 \Rightarrow \dots \Rightarrow u_k \Rightarrow v.$$

for some sequence  $u_1, u_2, \dots, u_k$ .

**Definition:** The language of the grammar is

$$\left\{ w \in \Sigma^* \mid S \xRightarrow{*} w \right\} .$$

# Example

Consider  $G_4 = (V, \{a, b\}, R, S)$ .

$R$  (Rules):  $S \rightarrow aSb \mid SS \mid \varepsilon$ .

Some words in the language:  $aabb$ ,  $aababb$ .

**Q.:** But what **is** this language?

**Hint:** Think of parentheses.

# Designing Context-Free Grammars

No recipe in general, but few rules-of-thumb

- If CFG is the **union** of several CFGs, rename variables (**not terminals**) so they are disjoint, and add new rule  $S \rightarrow S_1 \mid S_2 \mid \dots \mid S_i$ .
- For a regular language, grammar “follows” a DFA for the language. For initial state  $q_0$ , make  $R_0$  the start variable. For state transition  $\delta(q_i, a) = q_j$  add rule  $R_i \rightarrow aR_j$  to grammar. For each final state  $q_f$ , add rule  $R_f \rightarrow \varepsilon$  to grammar. This is called a **linear grammar**.
- For languages (like  $\{0^n \# 1^n \mid n \geq 0\}$ ), with **linked** substrings, a rule of form  $R \rightarrow uRv$  is helpful to force desired relation between substrings.

# Closure Properties

- **Regular languages** are closed under
  - union
  - concatenation
  - star
- **Context-Free Languages** are closed under
  - union :  $S \rightarrow S_1 \mid S_2$
  - concatenation  $S \rightarrow S_1 S_2$
  - star  $S_{new} \rightarrow \varepsilon \mid S_{old} \mid S_{new} S_{new}$

# More Closure Properties

- **Regular languages** are also closed under
  - complement (replace accept/non-accept states of DFA)
  - intersection ( $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ ).
- What about complement and intersection of **context-free languages**?
  - Absolutely not clear at this point (but will become crystal clear soon) . . .



# Ambiguity in Everyday's Language

“Woman without her man is nothing”

How would **you** parse and interpret this sentence?

# Ambiguity in Everyday's Language

“Woman without her man is nothing”

How would **you** parse and interpret this sentence?

**Option 1:** Woman, without her, man is nothing!

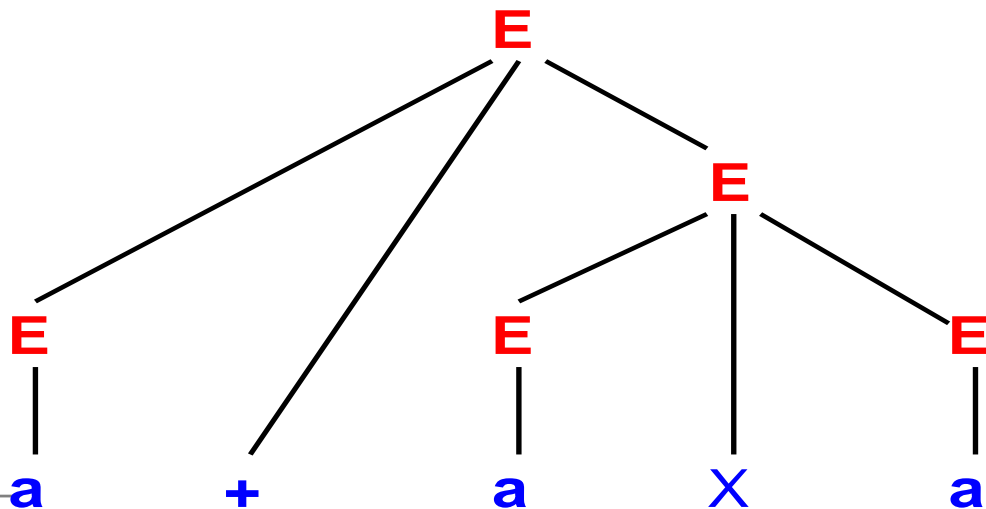
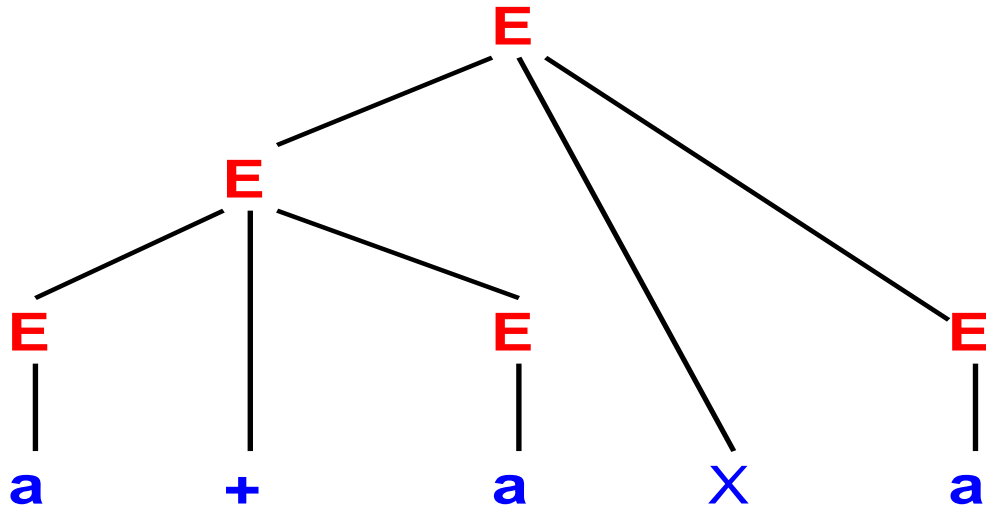
**Option 2:** Woman, without her man, is nothing!

Ambiguity in parsing (syntax) leads to ambiguity in meaning (semantics).

The same phenomena occurs in our beloved context free grammars.

# Ambiguity in CFLs

Grammar:  $E \rightarrow E + E \mid E \times E \mid (E) \mid a$



# Ambiguity

We say that a string  $w$  is derived **ambiguously** from grammar  $G$  if  $w$  has two or more parse trees that generate it from  $G$ .

Ambiguity is usually not only a syntactic notion but also **semantic**, implying multiple meanings for the same string. Think of  $a + a \times a$  from last grammar.

It is **sometime** possible to **eliminate** ambiguity by finding a different context free grammar generating the same language. This is true for the grammar above, which can essentially be replaced by the unambiguous grammar from lecture 3.

Some languages, e.g.  $\{1^i 2^j 3^k \mid i = j \text{ or } j = k\}$  are **inherently ambiguous**.

# Non-Context-Free Languages

- The **pumping lemma** for finite automata and **Myhill-Nerode** theorem are our tools for showing that languages are **not regular**.
- We will now show a similar **pumping lemma** for context-free languages.
- It is slightly more complicated . . .

# Pumping Lemma for CFL

Also known as the  $uvxyz$  Theorem.

**Theorem:** If  $A$  is a CFL, there is an  $\ell$  (critical length), such that if  $s \in A$  and  $|s| \geq \ell$ , then  $s = uvxyz$  where

- for every  $i \geq 0$ ,  $uv^i xy^i z \in A$
- $|vy| > 0$ , (non-triviality)
- $|vxy| \leq \ell$ .

# Basic Intuition

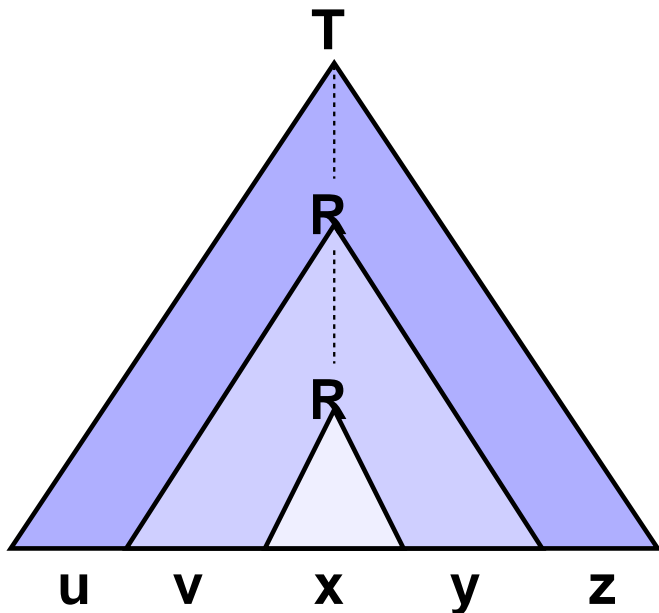
Let  $A$  be a CFL and  $G$  its CFG.

Let  $s$  be a “very long” string in  $A$ .

Then  $s$  must have a “tall” parse tree.

And some root-to-leaf path must repeat a symbol.

ahmmm,... why is that so?



# Proof

- let  $G$  be a CFG for CFL  $A$ .
- let  $b$  be the max number of symbols in right-hand-side of any rule.
- no node in parse tree has  $> b$  children.
- at depth  $d$ , can have at most  $b^d$  leaves.
- let  $|V|$  be the number of variables in  $G$ .
- set  $\ell = b^{|V|+2}$ .

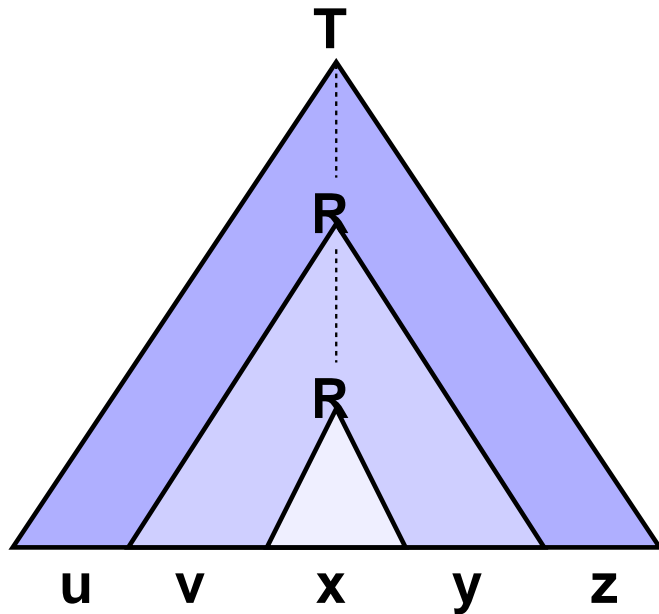


## Proof (2)

- let  $s$  be a string where  $|s| \geq \ell$
- let  $T$  be parse tree for  $s$  with fewest nodes
- $T$  has height  $\geq |V| + 2$
- some path in  $T$  has length  $\geq |V| + 2$
- that path **repeats** a variable  $R$

# Proof (3)

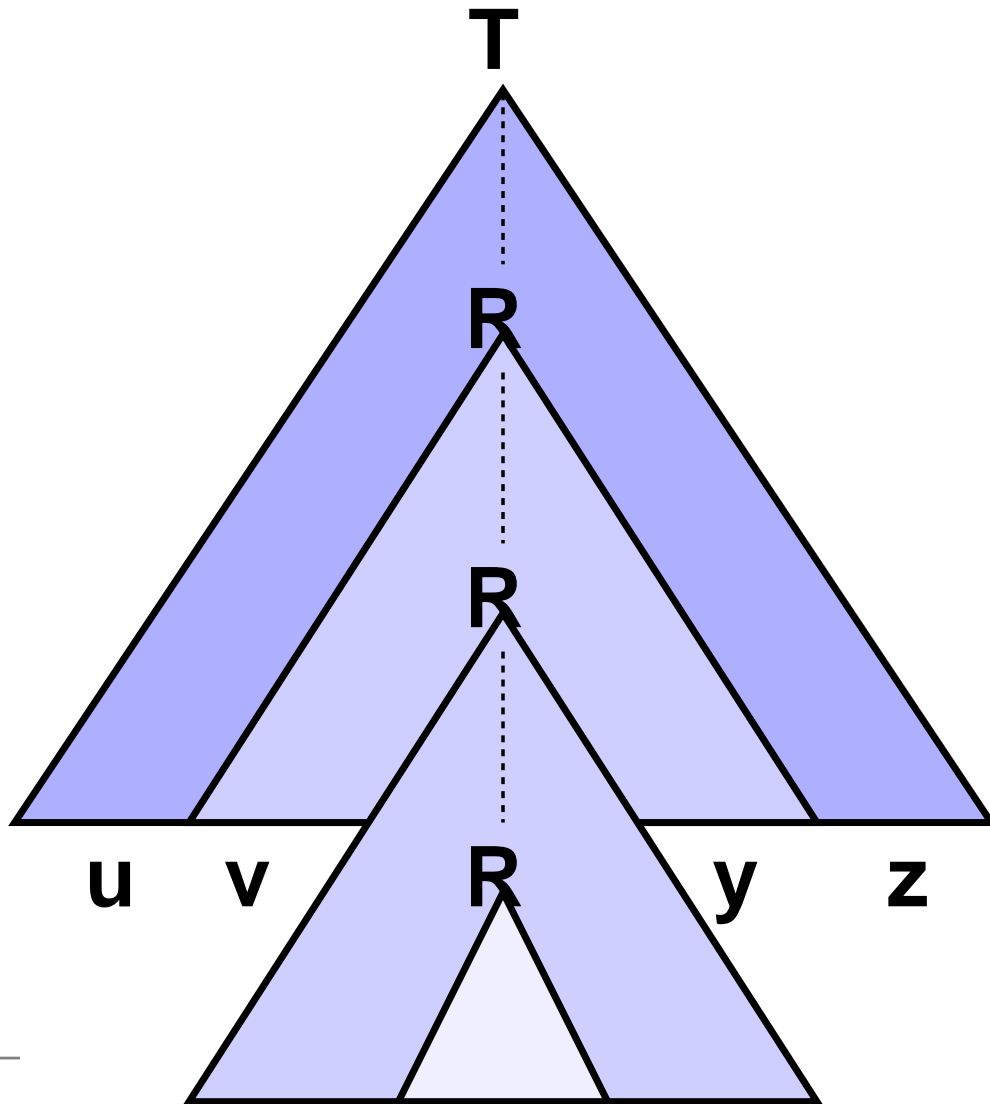
Split  $s = uvxyz$



- each occurrence of  $R$  produces a string
- upper produces string  $vxy$
- lower produces string  $x$

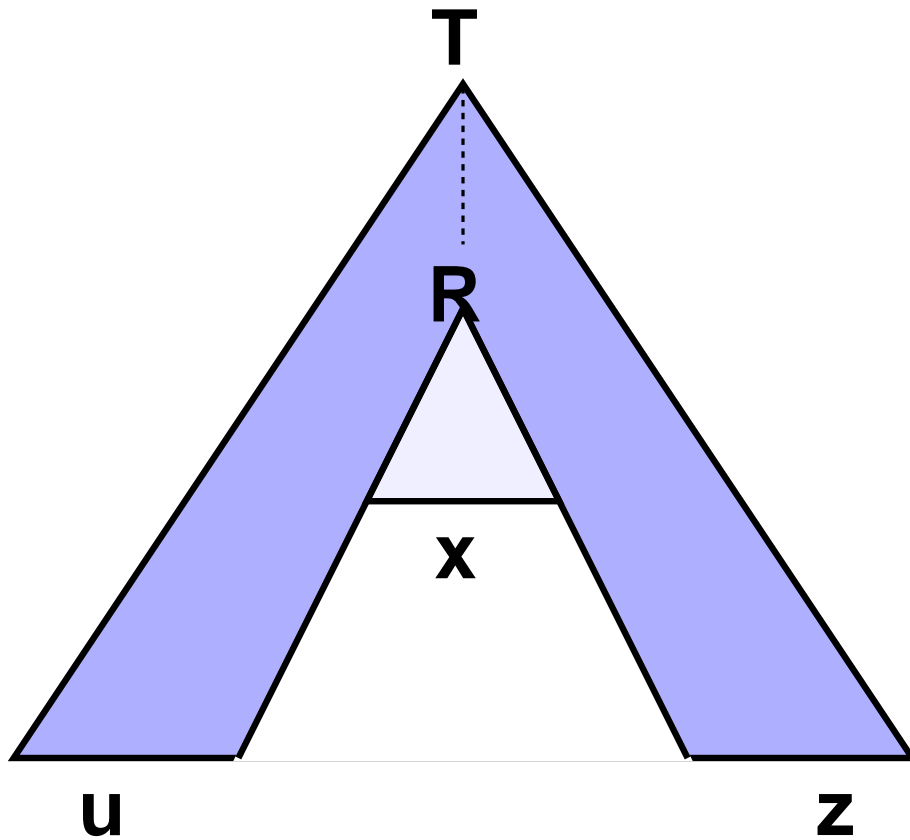
# Proof

Replacing smaller by larger yields  $uv^i xy^i z$ , for  $i > 0$ .



## Proof (4)

Replacing larger by smaller yields  $uxz$ .



Together, they establish:

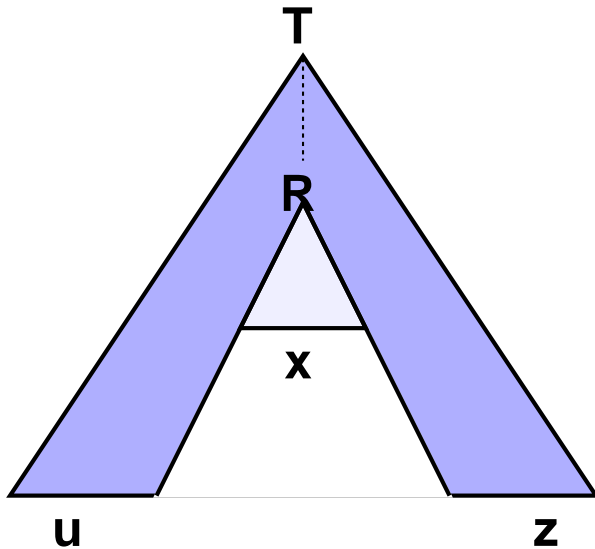
• for all  $i \geq 0$ ,  $uv^i xy^i z \in A$

## Proof (5)

Next condition is:

•  $|vy| > 0$  (out of  $uvwxyz$ )

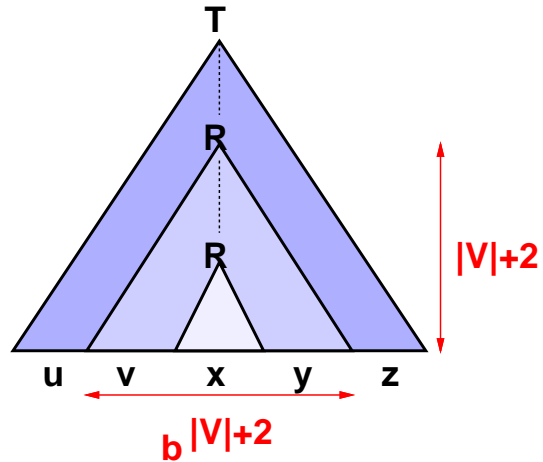
If  $v$  and  $y$  are both  $\varepsilon$ , then



is a parse tree for  $s$  with **fewer nodes** than  $T$ , contradiction.

# Proof (6)

Final condition is  $|vxy| \leq \ell$ :



- the upper occurrence of  $R$  generates  $vxy$ .
- can choose symbols such that both occurrences of  $R$  lie in bottom  $|V| + 1$  variables on path.
- subtree where  $R$  generates  $vxy$  is  $\leq |V| + 2$  high.
- strings by subtree at most  $b^{|V|+2} = \ell$  long. ♠

# Non CFL Example

**Theorem:**  $B = \{a^n b^n c^n\}$  is not a CFL.

**Proof:** By contradiction.

Let  $\ell$  be the critical length.

Consider  $s = a^\ell b^\ell c^\ell$ .

If  $s = uvxyz$ , neither  $v$  nor  $y$  can contain

- both  $a$ 's and  $b$ 's, or
- both  $b$ 's and  $c$ 's,

because otherwise  $uv^2xy^2z$  would have out-of-order symbols.

But if  $v$  and  $y$  contain only one letter, then  $uv^2xy^2z$  has an imbalance!

## Non CFL Example (2)

The language  $C = \{a^i b^j c^k \mid 0 \leq i \leq j \leq k\}$  is not context-free.

Let  $\ell$  be the critical length, and  $s = a^\ell b^\ell c^\ell$ .

Let  $s = uvxyz$

- neither  $v$  nor  $y$  contains two distinct symbols, because otherwise  $uv^2xy^2z$  would have out-of-order symbols.
- $vxy$  cannot be all  $b$ 's (why?)
- $|vxy| < \ell$ , so either
  - $v$  contains only  $a$ 's and  $y$  contains only  $b$ 's, but then  $uv^2xy^2z$  has too few  $c$ 's.
  - $v$  contains only  $b$ 's and  $y$  contains only  $c$ 's. but then  $uv^0xy^0z$  has too many  $a$ 's.



## Non CFL Example (3)

The language  $D = \{ww \mid w \in \{0, 1\}^*\}$  is not context-free.

Let  $s = 0^\ell 1^\ell 0^\ell 1^\ell$ . As before, suppose  $s = uvxyz$ .

Recall that  $|vxy| \leq \ell$ .

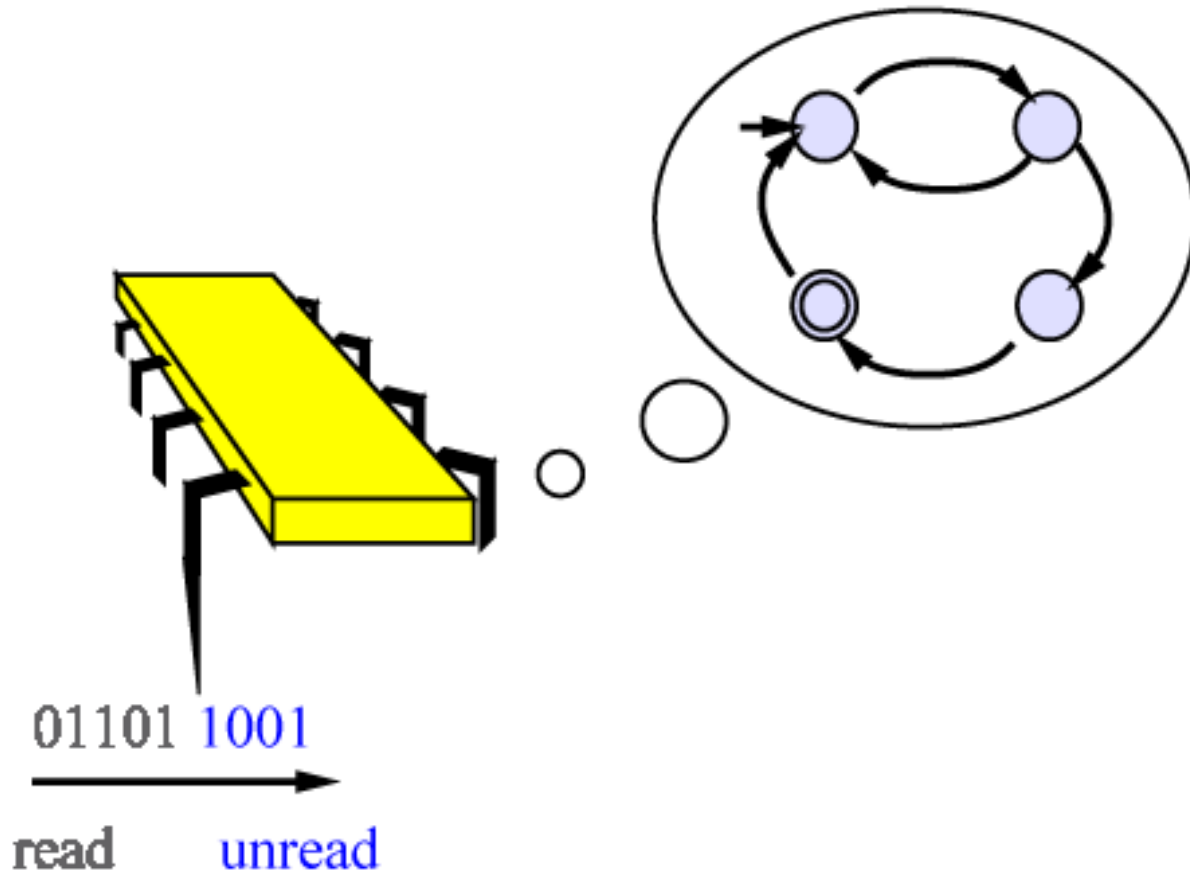
- if  $vxy$  is in the first half of  $s$ ,  $uv^2xy^2z$  moves a 1 into the first position in second half.
- if  $vxy$  is in the second half,  $uv^2xy^2z$  moves a 0 into the last position in first half.
- if  $vxy$  straddles the midpoint, then pumping *down* to  $uxz$  yields  $0^\ell 1^i 0^j 1^\ell$  where  $i$  and  $j$  cannot both be  $\ell$ .
- In fact, pumping *up* would also work in the last case.

Will shortly see that  $D = \{ww^R \mid w \in \{0, 1\}^*\}$  is CFL.

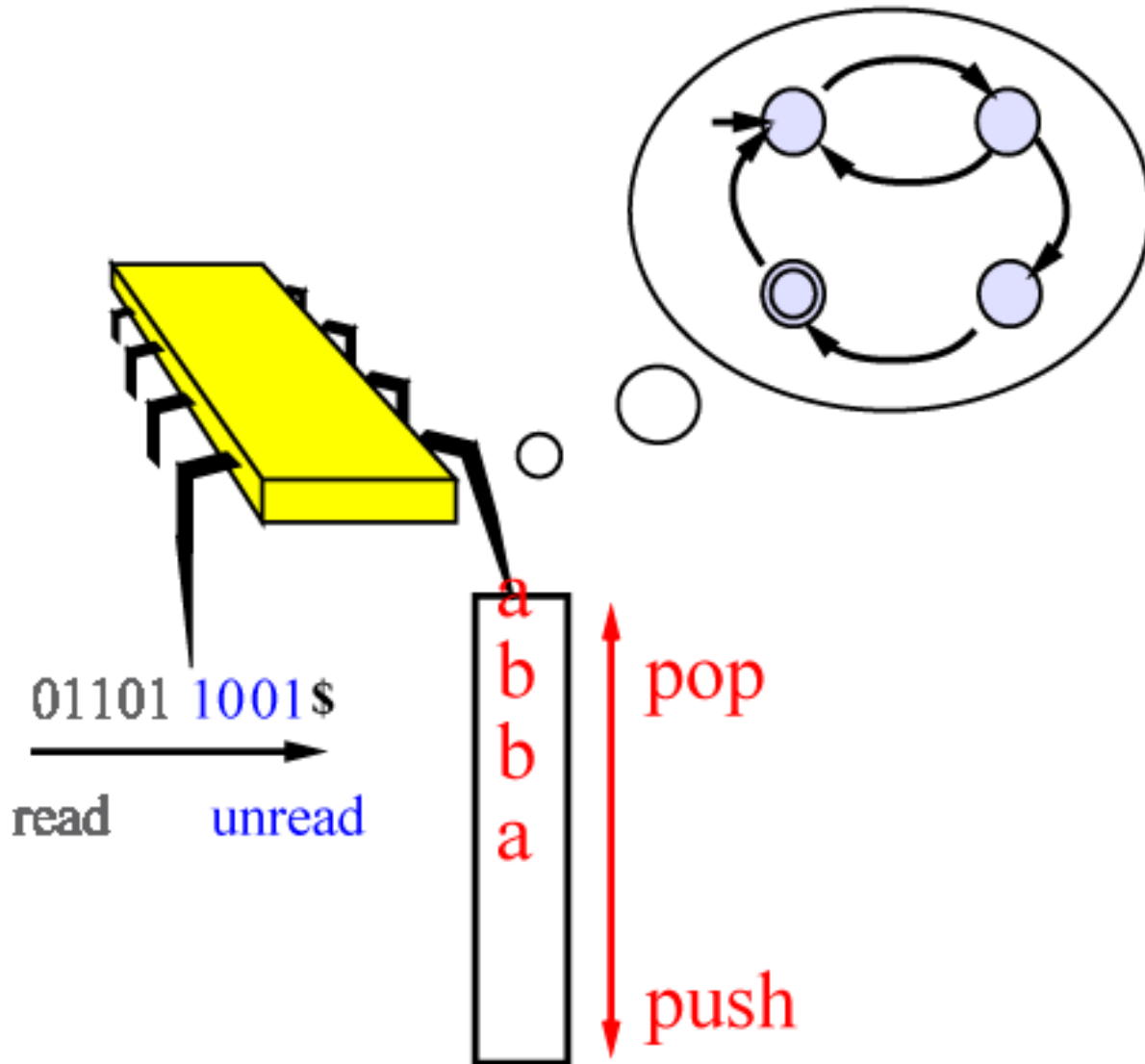
# String Generators and String Acceptors

- Regular expressions are **string generators** – they tell us how to generate all strings in a language  $L$
- Finite Automata (DFA, NFA) are **string acceptors** – they tell us if a specific string  $w$  is in  $L$
- CFGs are **string generators**
- Are there **string acceptors for** Context-Free languages?
- YES! Push-down automata

# A Finite Automaton



# A PushDown Automaton



# An Example

Recall that the language  $\{0^n 1^n \mid n \geq 0\}$  is not regular.  
Consider the following PDA:

- Read input symbols
- For each 0, push it on the stack
- As soon as a 1 is seen, pop a 0 for each 1 read
- **Accept** if stack is **empty** when **last symbol read**
- **Reject** if
  - Stack is **non-empty** when **end of input symbol read**
  - Stack is **empty** but **input symbol(s) still exist**,
  - 0 is read after 1.

# PDA Configuration

- A **Configuration** of a Push-Down Automata is a triplet
- ( $\langle \text{state} \rangle$ ,  $\langle \text{remaining input string} \rangle$ ,  $\langle \text{stack} \rangle$ ).
  
- A configuration is like a **snapshot** of PDA progress.
- A **PDA computation** is a sequence of successive configurations, starting from **start configuration**.
- The string describing the stack has top of the stack on the left (technical item).

# Comparing PDA and Finite Automata

- Our PDAs have a special **end of input** symbol,  $\$$ . This symbol is not part of input alphabet  $\Sigma$ . (this technical point **differs from book**, is not crucial, and just makes life easier.)
- PDA may be **deterministic** or **non-deterministic**.
- Unlike finite automata, non-determinism adds power: There are some languages accepted **only** by **non-deterministic** PDAs.

Transition function  $\delta$  looks different than DFA or NFA cases, reflecting **stack** functionality.

# PDA Transition Function

Denote input alphabet by  $\Sigma$  and stack alphabet by  $\Gamma$ .

- the **domain** of the transition function  $\delta$  is
  - current state:  $Q$
  - next input, if any:  $\Sigma_{\epsilon, \$}$  ( $=\Sigma \cup \{\epsilon\} \cup \{\$\}$ )
  - stack symbol popped, if any:  $\Gamma_{\epsilon}$  ( $=\Gamma \cup \{\epsilon\}$ )
- and its **range** is
  - new state:  $Q$
  - stack symbol pushed, if any:  $\Gamma_{\epsilon}$
  - non-determinism:  $\mathcal{P}$ (two components above)
- $\delta : Q \times \Sigma_{\epsilon, \$} \times \Gamma_{\epsilon} \rightarrow \mathcal{P}(Q \times \Gamma_{\epsilon})$



# Formal Definitions

A **pushdown automaton** (PDA) is a 6-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, F)$ , where

- $Q$  is a finite set called the *states*,
- $\Sigma$  is a finite set called the *input alphabet*,
- $\Gamma$  is a finite set called the *stack alphabet*,
- $\delta : Q \times \Sigma_{\epsilon, \$} \times \Gamma_{\epsilon} \rightarrow \mathcal{P}(Q \times \Gamma_{\epsilon})$  is the *transition function*,
- $q_0 \in Q$  is the *start state*, and
- $F \subseteq Q$  is the set of *accept states*.

# Conventions

- It will be convenient to be able to know when the **stack is empty**, but there is **no built-in mechanism** to do that.
  - Solution:
    - Start by pushing **\$** onto stack.
    - When you see it again, stack is empty.
  - **Question**: When is the input string exhausted?
    - **Answer**: When **\$** is read.
    - So PDA “knows” when the input terminates.

# Semi Formal Definitions

- A **pushdown automaton** (PDA)  $M$  accepts a string  $x$  if there is a computation of  $M$  on  $x$  (a sequence of state and stack transitions according to  $M$ 's transition function and corresponding to  $x$ ) that leads to an accepting state.
- The language accepted by  $M$  is the set of all strings  $x$  accepted by  $M$ .
- While a non-deterministic PDA,  $M$ , may have **many** computations on  $x$ , a deterministic PDA has just **one**.

# Notation

- Transition  $a, b \rightarrow c$  means
  - if read  $a$  from input
  - and pop  $b$  from stack
  - then push  $c$  onto stack
- Meaning of  $\varepsilon$  transitions:
  - if  $a = \varepsilon$ , don't read inputs
  - if  $b = \varepsilon$ , don't pop any symbols
  - if  $c = \varepsilon$ , don't push any symbols

# PDA Languages

The Push-Down Automata Languages,  $L_{PDA}$ , is the set of all languages that can be described by some PDA:

- $L_{PDA} = \{L : \exists \text{ PDA } M \wedge L[M] = L\}$

We already know  $L_{PDA} \supsetneq L_{DFA}$ , since every DFA is just a PDA that **ignores the stack**.

- $L_{CFG} \subseteq L_{PDA} ?$

- $L_{PDA} \subseteq L_{CFG} ?$

# Equivalence Theorem

**Theorem:** A language is context free if and only if some pushdown automaton accepts it.

- This time, both the “if” part and the “only if” part are non-trivial (unlike the proof of equivalence of regular languages and regular expression, where one part was trivial), and are somewhat tedious.
- We will later present a very high level view of the proof (not all details).
- Full details in pp. 106–114 of Sipser’s book, and/or slides from fall 2003/4.
- We will assume (and use) the theorem before proving it.
- For each of the following PDA languages, think of the CFGs that create them.

# PDA Transition Function – Reminder

Denote input alphabet by  $\Sigma$  and stack alphabet by  $\Gamma$ .

- the **domain** of the transition function  $\delta$  is
  - current state:  $Q$
  - next input, if any:  $\Sigma_{\varepsilon, \$}$  ( $=\Sigma \cup \{\varepsilon\} \cup \{\$\}$ )
  - stack symbol popped, if any:  $\Gamma_{\varepsilon}$  ( $=\Gamma \cup \{\varepsilon\}$ )
- and its **range** is
  - new state:  $Q$
  - stack symbol pushed, if any:  $\Gamma_{\varepsilon}$
  - non-determinism:  $\mathcal{P}$ (two components above)
- $\delta : Q \times \Sigma_{\varepsilon, \$} \times \Gamma_{\varepsilon} \rightarrow \mathcal{P}(Q \times \Gamma_{\varepsilon})$

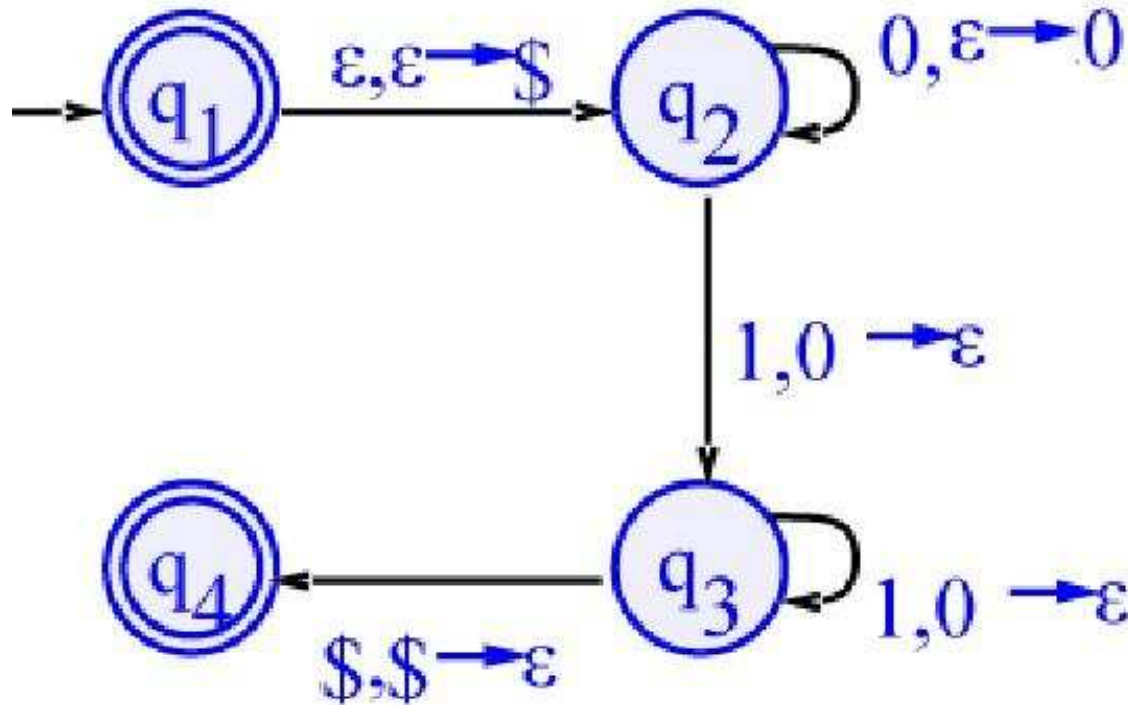
# The Transition Function for Deterministic PDAs

- the **domain** of the transition function  $\delta$  is
  - current state:  $Q$
  - next input, if any:  $\Sigma_{\varepsilon, \$}$  ( $=\Sigma \cup \{\varepsilon\} \cup \{\$\}$ )
  - stack symbol popped, if any:  $\Gamma_{\varepsilon}$  ( $=\Gamma \cup \{\varepsilon\}$ )
- and its **range** is
  - next state and stack symbol pushed, if any:  $Q \times \Gamma_{\varepsilon}$
  - or **nothing** ( $\emptyset$ ), meaning no continuation.
- $\delta : Q \times \Sigma_{\varepsilon, \$} \times \Gamma_{\varepsilon} \rightarrow (Q \times \Gamma_{\varepsilon}) \cup \{\emptyset\}$
- $\forall q \in Q, a \in \Sigma_{\$}, \gamma \in \Gamma_{\varepsilon}$ , if  $\delta(q, \varepsilon, \gamma) \neq \emptyset$ , then  $\delta(q, a, \gamma) = \emptyset$   
(epsilon input transition prevents “real” input transition).
- $\forall q \in Q, a \in \Sigma_{\varepsilon, \$}, \gamma \in \Gamma$ , if  $\delta(q, a, \varepsilon) \neq \emptyset$ , then  $\delta(q, a, \gamma) = \emptyset$   
(epsilon stack transition prevents “real” stack transition).



# Example: Deterministic PDA

The PDA



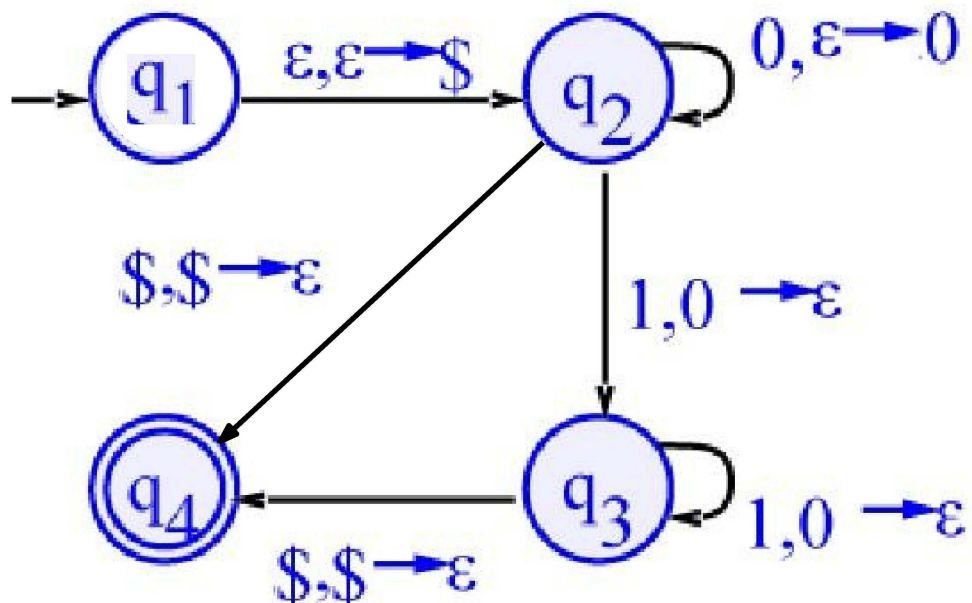
accepts  $\{0^n 1^n \mid n \geq 1\}$ .

Does it also accept the empty string?

If not, how would you fix it?

# Example – Modified to Accept $\varepsilon$

The PDA



now accepts  $\{0^n 1^n \mid n \geq 0\}$ .

# Another Example

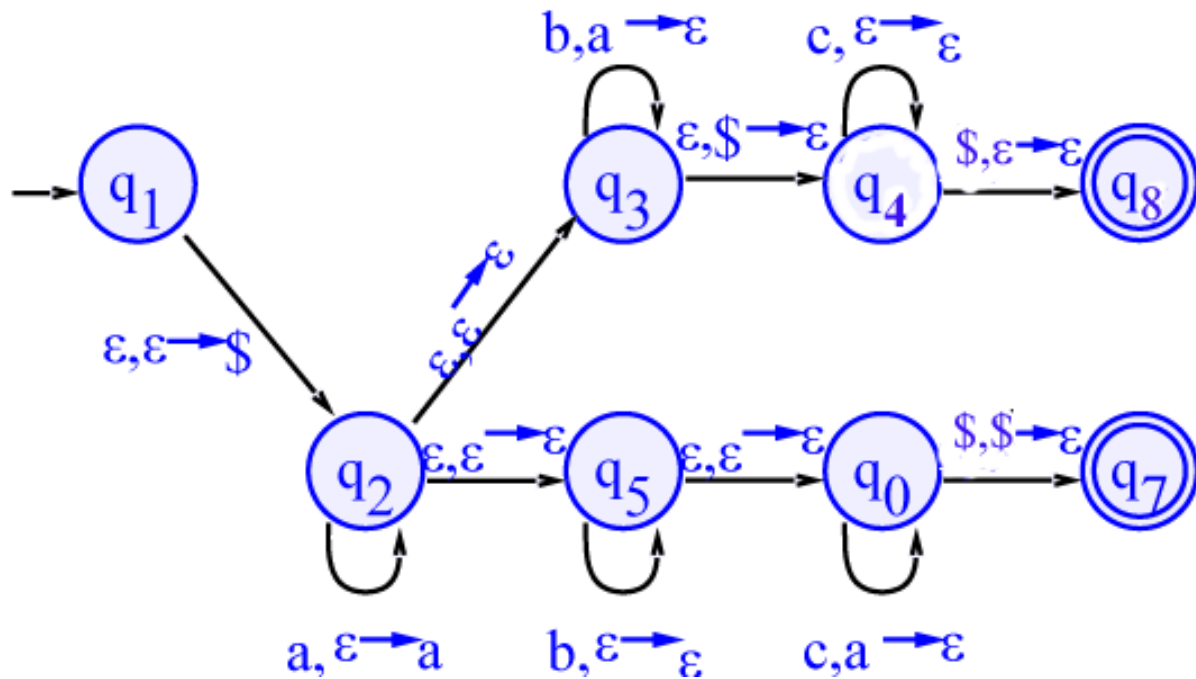
A PDA that accepts

$$\left\{ a^i b^j c^k \mid i, j, k > 0 \text{ and } i = j \text{ or } i = k \right\}$$

Informally:

- read and push  $a$ 's
- either pop and match with  $b$ 's
- or else pop and match with  $c$ 's
- a **non-deterministic** choice!

## Another Example (cont.)



This PDA accepts

$$\left\{ a^i b^j c^k \mid i, j, k > 0 \text{ and } i = j \text{ or } i = k \right\}$$

## Another Example (cont.)

A PDA that accepts  $\{a^i b^j c^k \mid i, j, k > 0 \text{ and } i = j \text{ or } i = k\}$

- **Note:** non-determinism is essential here!
- Unlike finite automata, non-determinism **does add power.**
- Let us try to think how a proof could go,
- ⋮
- and realize it does not seem trivial or immediate.
- We will later give a proof that the language  $L = \{x^n y^n \mid n \geq 0\} \cup \{x^n y^{2n} \mid n \geq 0\}$  is accepted by a non-deterministic PDA but **not** by a deterministic one.

# Another PDA Example

A **palindrome** is a string  $w$  satisfying  $w = w^{\mathcal{R}}$ .

- “Madam I’m Adam”
- “Dennis and Edna sinned”
- “Red rum, sir, is murder”
- “Able was I ere I saw Elba”
- “In girum imus nocte et consumimur igni” (Latin: “we go into the circle by night, we are consumed by fire”.)
- “*νιψον ανομηματα μη μοναν οψιν*”
- Palindromes also appear in nature. For example as DNA **restriction sites** – short genomic strings over  $\{A, C, T, G\}$ , being cut by (naturally occurring) **restriction enzymes**.

# A PDA that Recognizes Palindromes

- On input  $x$ , the PDA starts pushing  $x$  into stack.
- At some point, PDA **guesses** that the mid point of  $x$  was reached.
- Pops and compares to input, letter by letter.
- If end of input occurs together with emptying of stack, accept.
- This PDA accepts palindromes of ***even length*** over the alphabet (all lengths is easy modification).
- Again, non-determinism (at which point to make the switch) **seems** necessary.
- Q.: Which **CFG** generates palindromes?

# Chomsky Normal Form

A simplified, canonical form of context free grammars. Elegant by itself, useful (but not crucial) in proving equivalence theorem. Can also be used to slightly simplify proof of pumping lemma. Every rule has the form

$$A \rightarrow BC$$

$$A \rightarrow a$$

$$S \rightarrow \varepsilon$$

where  $S$  is the start symbol,  $A$ ,  $B$  and  $C$  are any variable, except  $B$  and  $C$  not the start symbol, and  $A$  can be the start symbol.



# Theorem

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

Basic idea:

- Add new start symbol  $S_0$ .
- Eliminate all  $\varepsilon$  rules of the form  $A \rightarrow \varepsilon$ .
- Eliminate all “unit” rules of the form  $A \rightarrow B$ .
- Patch up rules so that grammar generates the same language.
- Convert remaining “long rules” to proper form.

# Proof

Add new start symbol  $S_0$  and rule  $S_0 \rightarrow S$ .

Guarantees that new start symbol does not appear on right-hand-side of a rule.

# Proof

Eliminating  $\varepsilon$  rules.

Repeat:

- remove some  $A \rightarrow \varepsilon$ .
- for each  $R \rightarrow uAv$ , add rule  $R \rightarrow uv$ .
- and so on: for  $R \rightarrow uAvAw$  add  $R \rightarrow uvAw$ ,  $R \rightarrow uAvw$ , and  $R \rightarrow uvw$ .
- for  $R \rightarrow A$  add  $R \rightarrow \varepsilon$ , except if  $R \rightarrow \varepsilon$  has already been removed.

until all  $\varepsilon$ -rules not involving the original start variable have been removed.

# Proof

Eliminate unit rules.

Repeat:

- remove some  $A \rightarrow B$ .
- for each  $B \rightarrow u$ , add rule  $A \rightarrow u$ , unless this is previously removed unit rule. ( $u$  is a string of variables and terminals.)

until all unit rules have been removed.

# Proof

Finally, convert long rules.

To replace each  $A \rightarrow u_1 u_2 \dots u_k$  (for  $k \geq 3$ ), introduce new non-terminals

$$N_1, N_2, \dots, N_{k-1}$$

and rules

$$A \rightarrow u_1 N_1$$

$$N_1 \rightarrow u_2 N_2$$

$$\vdots$$

$$N_{k-3} \rightarrow u_{k-2} N_{k-2}$$

$$N_{k-2} \rightarrow u_{k-1} u_k$$



# Conversion Example

Initial Grammar:

$$S \rightarrow ASA \mid aB$$

$$A \rightarrow B \mid S$$

$$B \rightarrow b \mid \varepsilon$$

(1) Add new start state:

$$S_0 \rightarrow S$$

$$S \rightarrow ASA \mid aB$$

$$A \rightarrow B \mid S$$

$$B \rightarrow b \mid \varepsilon$$

## Conversion Example (2)

$$\begin{aligned}S_0 &\rightarrow S \\S &\rightarrow ASA \mid aB \\A &\rightarrow B \mid S \\B &\rightarrow b \mid \varepsilon\end{aligned}$$

(2) Remove  $\varepsilon$ -rule  $B \rightarrow \varepsilon$ :

$$\begin{aligned}S_0 &\rightarrow S \\S &\rightarrow ASA \mid aB \mid a \\A &\rightarrow B \mid S \mid \varepsilon \\B &\rightarrow b \mid \varepsilon\end{aligned}$$

## Conversion Example (3)

$$\begin{aligned}S_0 &\rightarrow S \\S &\rightarrow ASA \mid aB \mid a \\A &\rightarrow B \mid S \mid \varepsilon \\B &\rightarrow b\end{aligned}$$

(3) Remove  $\varepsilon$ -rule  $A \rightarrow \varepsilon$ :

$$\begin{aligned}S_0 &\rightarrow S \\S &\rightarrow ASA \mid aB \mid a \mid AS \mid SA \mid S \\A &\rightarrow B \mid S \mid \varepsilon \\B &\rightarrow b\end{aligned}$$



## Conversion Example (4)

$$S_0 \rightarrow S$$

$$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA \mid S$$

$$A \rightarrow B \mid S$$

$$B \rightarrow b$$

(4) Remove unit rule  $S \rightarrow S$

$$S_0 \rightarrow S$$

$$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA \mid S$$

$$A \rightarrow B \mid S$$

$$B \rightarrow b$$

## Conversion Example (5)

$$S_0 \rightarrow S$$

$$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$$

$$A \rightarrow B \mid S$$

$$B \rightarrow b$$

(5) Remove unit rule  $S_0 \rightarrow S$ :

$$S_0 \rightarrow S \mid ASA \mid aB \mid a \mid AS \mid SA$$

$$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$$

$$A \rightarrow B \mid S$$

$$B \rightarrow b$$

## Conversion Example (6)

$$S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$$

$$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$$

$$A \rightarrow B \mid S$$

$$B \rightarrow b$$

(6) Remove unit rule  $A \rightarrow B$ :

$$S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$$

$$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$$

$$A \rightarrow B \mid S \mid b$$

$$B \rightarrow b$$

## Conversion Example (7)

$$S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$$

$$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$$

$$A \rightarrow S \mid b$$

$$B \rightarrow b$$

Remove unit rule  $A \rightarrow S$ :

$$S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$$

$$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$$

$$A \rightarrow S \mid b \mid ASA \mid aB \mid a \mid AS \mid SA$$

$$B \rightarrow b$$

## Conversion Example (8)

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

(8) Final simplification – treat **long rules**:

$$S_0 \rightarrow AA_1 \mid UB \mid a \mid SA \mid AS$$
$$S \rightarrow AA_1 \mid UB \mid a \mid SA \mid AS$$
$$A \rightarrow b \mid AA_1 \mid UB \mid a \mid SA \mid AS$$
$$A_1 \rightarrow SA$$
$$U \rightarrow a$$
$$B \rightarrow b$$
