

# CS244: THEORY OF COMPUTATION

Fu Song  
ShanghaiTech University

Fall 2020

## Context-free Grammar and Pushdown Automata

# Outline

## Context-free Grammar and Pushdown Automata

- Context-free Grammar

- Ambiguity

- Chomsky Normal Form

- Pushdown Automata

- Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

- Pumping lemma

- Closure Properties

## Deterministic Context-Free Languages

- Deterministic Context-Free Languages

- Properties of DCFLs

- Deterministic context-free grammars

- Relationship of DPDAs and DCFGs

# Outline

## Context-free Grammar and Pushdown Automata

### Context-free Grammar

Ambiguity

Chomsky Normal Form

Pushdown Automata

Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

Pumping lemma

Closure Properties

## Deterministic Context-Free Languages

Deterministic Context-Free Languages

Properties of DCFLs

Deterministic context-free grammars

Relationship of DPDAs and DCFGs

# Definition

A **Context-free Grammar** CFG is a formal grammar  $(\mathcal{N}, \Sigma, \mathcal{P}, S)$  where

1.  $\mathcal{N}$  is a finite set of non-terminals,
2.  $\Sigma$  is a finite set of terminals, disjoint from  $\mathcal{N}$ ,
3.  $\mathcal{P}$  is a finite set of production **rules**, of the form

$$A \rightarrow \alpha$$

with non-terminal  $A \in \mathcal{N}$  and  $\alpha \in (\mathcal{N} \cup \Sigma)^*$ .

4.  $S \in \mathcal{N}$  is the **start symbolic**.

Extended Backus-Naur form (EBNF) is a family of metasyntax notations, any of which can be used to express a context-free grammar

Example: CFG for arithmetic expressions,

$$\begin{array}{ll} E \rightarrow E + E & E \rightarrow E * E \\ E \rightarrow (E) & E \rightarrow id \end{array}$$

# Derivations

Let  $\alpha, \beta, \gamma$  be strings of non-terminals and terminals, and  $A \rightarrow \gamma$  is a rule of the grammar. We say that  $\alpha A \beta$  yields  $\alpha \gamma \beta$ , written

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

Say that  $\alpha$  derives  $\beta$ , written  $\alpha \xRightarrow{*} \beta$ , if  $\alpha = \beta$  or if a sequence  $\gamma_1, \gamma_2, \dots, \gamma_k$  exists for  $k \geq 0$  and

$$\alpha \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \dots \Rightarrow \gamma_k \Rightarrow \beta.$$

The language  $L(G)$  of the grammar  $G$  is

$$\{\alpha \in \Sigma^* \mid S \xRightarrow{*} \alpha\},$$

which is a context-free language (CFL).

## Example

Recalling the CFG for arithmetic expressions,

$$\begin{array}{ll} E \rightarrow E + E & E \rightarrow E * E \\ E \rightarrow (E) & E \rightarrow id \end{array}$$

$$E \Rightarrow E * E \Rightarrow (E) * E \Rightarrow (E + E) * E \Rightarrow (id + id) * id.$$

## More Examples

1. The language  $\{0^n 1^n \mid n \geq 0\}$  has a grammar:

$$S_1 \rightarrow 0S_11 \mid \varepsilon.$$

2. The language  $\{1^n 0^n \mid n \geq 0\}$  has a grammar:

$$S_2 \rightarrow 1S_20 \mid \varepsilon.$$

3. The language

$$\{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$$

has a grammar:

$$S \rightarrow S_1 \mid S_2$$

$$S_1 \rightarrow 0S_11 \mid \varepsilon$$

$$S_2 \rightarrow 1S_20 \mid \varepsilon.$$



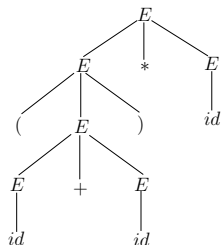
# Derivation Trees

A **derivation tree** for a CFG  $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$  is a **labeled tree** s.t.

- ▶ every vertex has a label from  $\mathcal{N} \cup \Sigma \cup \{\varepsilon\}$ ,
- ▶ the root is labeled by  $S$ ,
- ▶ every internal vertex is labeled by symbols from  $\mathcal{N}$ ,
- ▶ for every vertex labeled by  $A \in \mathcal{N}$ ,  
if its children are labeled by  $X_1, \dots, X_k$  (from left to right),  
then  $A \rightarrow X_1 \dots X_k$  is a rule of  $G$ .

Example: Derivation tree for  $(id + id) * id$

$$\begin{array}{ll} E \rightarrow E + E & E \rightarrow E * E \\ E \rightarrow (E) & E \rightarrow id \end{array}$$



# Leftmost Derivation and Rightmost Derivation

**Leftmost** derivation:

*Each step in the derivation is applied to the **leftmost** nonterminal.*

Roughly speaking, **leftmost** derivation corresponds to

*a **preorder** DFS traversal of the derivation tree.*

**Rightmost** derivation:

*Each step in the derivation is applied to the **rightmost** nonterminal.*

Roughly speaking, **rightmost** derivation corresponds to

*a **postorder** DFS traversal of the derivation tree.*

# Leftmost Derivation Example

$E$

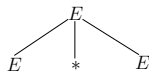
Example: Derivation tree for  $(id + id) * id$

$$\begin{array}{ll} E \rightarrow E + E & E \rightarrow E * E \\ E \rightarrow (E) & E \rightarrow id \end{array}$$

# Leftmost Derivation Example

Example: Derivation tree for  $(id + id) * id$

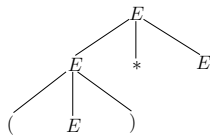
$$\begin{array}{ll} E \rightarrow E + E & E \rightarrow E * E \\ E \rightarrow (E) & E \rightarrow id \end{array}$$



# Leftmost Derivation Example

Example: Derivation tree for  $(id + id) * id$

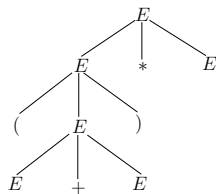
$$\begin{array}{ll} E \rightarrow E + E & E \rightarrow E * E \\ E \rightarrow (E) & E \rightarrow id \end{array}$$



# Leftmost Derivation Example

Example: Derivation tree for  $(id + id) * id$

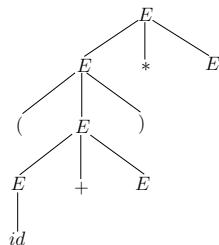
$$\begin{array}{ll} E \rightarrow E + E & E \rightarrow E * E \\ E \rightarrow (E) & E \rightarrow id \end{array}$$



# Leftmost Derivation Example

Example: Derivation tree for  $(id + id) * id$

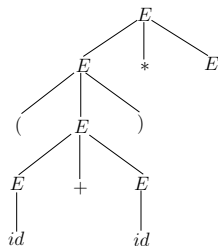
$$\begin{array}{ll} E \rightarrow E + E & E \rightarrow E * E \\ E \rightarrow (E) & E \rightarrow id \end{array}$$



# Leftmost Derivation Example

Example: Derivation tree for  $(id + id) * id$

$$\begin{array}{ll} E \rightarrow E + E & E \rightarrow E * E \\ E \rightarrow (E) & E \rightarrow id \end{array}$$

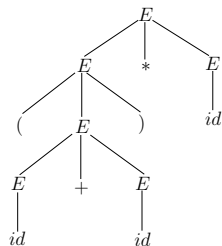




# Leftmost Derivation Example

Example: Derivation tree for  $(id + id) * id$

$$\begin{array}{ll} E \rightarrow E + E & E \rightarrow E * E \\ E \rightarrow (E) & E \rightarrow id \end{array}$$



# Outline

## Context-free Grammar and Pushdown Automata

- Context-free Grammar

- Ambiguity**

- Chomsky Normal Form

- Pushdown Automata

- Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

- Pumping lemma

- Closure Properties

## Deterministic Context-Free Languages

- Deterministic Context-Free Languages

- Properties of DCFLs

- Deterministic context-free grammars

- Relationship of DPDAs and DCFGs

$$\begin{array}{ll}
 E \rightarrow E + E & E \rightarrow E * E \\
 E \rightarrow (E) & E \rightarrow id
 \end{array}$$

The string  $(id + id) * id$  have two different derivations:

1.  $\langle E \rangle \rightarrow \langle E \rangle * \langle E \rangle \Rightarrow \langle E \rangle + \langle E \rangle * \langle E \rangle \xRightarrow{*} (id + id) * id.$
2.  $\langle E \rangle \rightarrow \langle E \rangle + \langle E \rangle \Rightarrow \langle E \rangle + \langle E \rangle * \langle E \rangle \xRightarrow{*} (id + id) * id.$

# Ambiguity

## Definition

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.

$\{a\}$  has two different grammars

1.  $S \rightarrow S' \mid a; S' \rightarrow a$
2.  $S \rightarrow a$

The first is ambiguous, while the second is not.

$$\{a^i b^j c^k \mid i = j \text{ or } j = k\}$$

is **inherently ambiguous**, i.e., its every grammar is ambiguous.

# Outline

## Context-free Grammar and Pushdown Automata

- Context-free Grammar

- Ambiguity

- Chomsky Normal Form**

- Pushdown Automata

- Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

- Pumping lemma

- Closure Properties

## Deterministic Context-Free Languages

- Deterministic Context-Free Languages

- Properties of DCFLs

- Deterministic context-free grammars

- Relationship of DPDAs and DCFGs

# Chomsky Normal Form

## Definition

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

$$A \rightarrow BC$$

$$A \rightarrow a$$

where  $a$  is any terminal and  $A$ ,  $B$ , and  $C$  are any non-terminals – except that  $B$  and  $C$  may be not the start variable.

In addition, we permit the rule  $S \rightarrow \varepsilon$ , where  $S$  is the start variable.

## Theorem

*For every CFG  $G$ , a Chomsky normal form CFG  $G'$  can be computed in linear time such that  $L(G) = L(G')$ .*

# Proof

1. Add a **new start non-terminal**  $S_0$  with the rule  $S_0 \rightarrow S$ , where  $S$  is the original start variable.
2. **Remove every**  $A \rightarrow \varepsilon$ , where  $A \neq S$ . Then for each occurrence of  $A$  on the right-hand side of a rule, we add a new rule with that occurrence deleted. For  $R \rightarrow A$ , we add  $R \rightarrow \varepsilon$  unless we had previously removed  $R \rightarrow \varepsilon$ .
3. **Remove every**  $A \rightarrow B$ . Then whenever a rule  $B \rightarrow \alpha$  appears, where  $\alpha$  is a string of non-terminal and terminals, we add the rule  $A \rightarrow \alpha$  unless this was previously removed.
4. **Replace each rule**  $A \rightarrow \sigma_1 \sigma_2 \cdots \sigma_k$  with  $k \geq 3$  and each  $\sigma_i$  is a non-terminal or terminal with the rules

$$A \rightarrow \sigma_1 A_1, A_1 \rightarrow \sigma_2 A_2, A_2 \rightarrow \sigma_3 A_3, \dots, \text{ and } A_{k-2} \rightarrow \sigma_{k-1} \sigma_k.$$

The  $A_i$ 's are **new non-terminals**. We replace any terminal  $\sigma_i$  with the new variable  $C_i$  and add  $C_i \rightarrow \sigma_i$ .

# Outline

## Context-free Grammar and Pushdown Automata

- Context-free Grammar

- Ambiguity

- Chomsky Normal Form

- Pushdown Automata**

- Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

- Pumping lemma

- Closure Properties

## Deterministic Context-Free Languages

- Deterministic Context-Free Languages

- Properties of DCFLs

- Deterministic context-free grammars

- Relationship of DPDAs and DCFGs



# Pushdown automata

## Definition

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

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

# Configuration

## Configuration

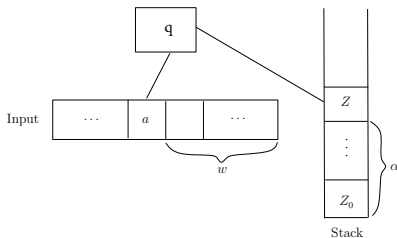
Given a PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ , a **configuration** is a tuple  $(q, w, \gamma)$  such that

- ▶  $q \in Q$ ,
- ▶  $w \in \Sigma^*$  (the suffix of the input not read yet),
- ▶  $\gamma \in \Gamma^*$  (the content of the stack, with the topmost symbol leftmost).

## $\vdash_M$ Relation

The  $\vdash_M$  relation between configurations is defined as follows:

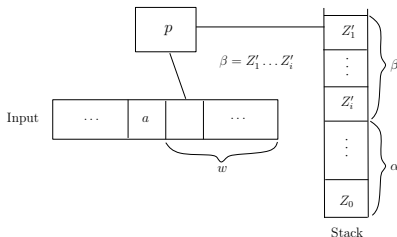
If  $(p, \beta) \in \delta(q, a, Z)$  (where  $a \in \Sigma_\varepsilon$ ), then  $(q, aw, Z\alpha) \vdash_M (p, w, \beta\alpha)$ .



# $\vdash_M$ Relation

The  $\vdash_M$  relation between configurations is defined as follows:

If  $(p, \beta) \in \delta(q, a, Z)$  (where  $a \in \Sigma_\varepsilon$ ), then  $(q, aw, Z\alpha) \vdash_M (p, w, \beta\alpha)$ .



Let  $\vdash_M^*$  be reflexive and transitive relation of  $\vdash_M^*$ , namely,

- ▶  $(q, w, \gamma) \vdash_M^* (q, w, \gamma)$  for each configuration  $(q, w, \gamma)$ ;
- ▶  $(q, w, \gamma) \vdash_M^* (q'', w'', \gamma'')$  if  $(q, w, \gamma) \vdash_M^* (q', w', \gamma')$  and  $(q', w', \gamma') \vdash_M (q'', w'', \gamma'')$

# Accepted Languages

Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  be a PDA.

- Define  $L(M)$ , the language accepted by **accepting state**, as follows,

$$\{w \in \Sigma^* \mid (q_0, w, \epsilon) \vdash_M^* (p, \epsilon, \gamma) \text{ for } p \in F, \gamma \in \Gamma^*\}.$$

## PDA for $\{0^n 1^n \mid n \geq 0\}$

PDA  $M = (Q, \Sigma, \Gamma, \delta, q_1, F)$  accepts  $\{0^n 1^n \mid n \geq 0\}$  by accepting states

$$Q = \{q_1, q_2, q_3, q_4\},$$

$$\Sigma = \{0, 1\},$$

$$\Gamma = \{0, \$\},$$

$$F = \{q_1, q_4\}, \text{ and}$$

$\delta$  is given by the following table, wherein blank entries signify  $\emptyset$ .

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

# Pushdown automata with start stack symbol

## Definition

A **pushdown automaton** (PDA)  $M$  is a 7-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ , where

1.  $Q$  is a finite set of states,
  2.  $\Sigma$  is a finite set of input alphabet,
  3.  $\Gamma$  is a finite set of stack alphabet,
  4.  $\delta : Q \times \Sigma_{\epsilon} \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$  is the transition function,
  5.  $q_0 \in Q$  is the start state,
  6.  $Z_0 \in \Gamma$  is a start stack symbol
  7.  $F \subseteq Q$  is the set of accept states.
- Define  $L(M)$ , the language accepted by **accepting state**, as follows,

$$\{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_M^* (p, \epsilon, \gamma) \text{ for } p \in F, \gamma \in \Gamma^*\}.$$

- Define  $N(M)$ , the language accepted by **empty stack**, as follows,

$$\{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_M^* (p, \epsilon, \epsilon) \text{ for } p \in Q\}.$$

## Quiz

Define a PDA  $M = (\{q_0, q_1\}, \{0, 1\}, \{A, B\}, \delta, q_0, Z_0, \emptyset)$  accepts  $\{ww^R \mid w \in \{0, 1\}^*\}$  by empty stack.



# Eliminating Start Stack Symbol

Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  be a PDA with start stack symbol. The goal: Define a PDA without start stack symbol  $M'$  such that  $L(M') = L(M)$ . The intuition:

*$M'$  starts from a new state  $q'_0$ , pushes  $Z_0$  onto the stack, and goes to  $q_0$ .*

$M' = (Q \cup \{q'_0\}, \Sigma, \Gamma, \delta', q'_0, F)$  is defined as follows.  $\delta'$  includes all the elements of  $\delta$  and the transition

- ▶  $\delta'(q'_0, \varepsilon, \varepsilon) = \{(q_0, Z_0)\}$ .

# Adding Start Stack Symbol

Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  be a PDA without start stack symbol.

The goal: Define a PDA with start stack symbol  $M'$  such that

$L(M') = L(M)$ . The intuition:

*Add a new start symbol  $Z_0$  to  $M$  which will not be popped from the stack, and enumerates all the possible topmost stack symbols for  $\delta(q, \sigma, \varepsilon)$ .*

$M' = (Q \cup \{q'_0\}, \Sigma, \Gamma \cup \{Z_0\}, \delta', q'_0, Z_0, F)$  is defined as follows.

$\delta'$  includes the transitions

- ▶  $\delta'(q'_0, \varepsilon, Z_0) = \{(q_0, Z_0)\}$
- ▶  $\delta'(q, \sigma, Z) = \{(q, \gamma Z) \mid (q, \gamma) \in \delta(q, \sigma, \varepsilon), Z \in \Gamma \cup \{Z_0\}\}$
- ▶  $\delta'(q, \sigma, Z) = \delta(q, \sigma, Z)$

## From accepting state to empty stack

Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  be a PDA.

The goal: Define a PDA  $M'$  such that  $N(M') = L(M)$ .

The intuition:

*if  $M$  accepts an input by a final state  $q$ , then from  $q$ ,  $M'$  enters a new state  $q_e$ , empties the stack, and accepts.*

In addition,  $M'$  includes a new start symbol  $Z'_0$  to deal with the following situation,

*$M$  reads all the input, enters a non-final state, and the stack is empty.*

$M' = (Q \cup \{q'_0, q_e\}, \Sigma, \Gamma \cup \{Z'_0\}, \delta', q'_0, Z'_0, \emptyset)$  is defined as follows.

$\delta'$  includes all the elements of  $\delta$  and the transitions

- ▶  $\delta'(q'_0, \varepsilon, Z'_0) = \{(q_0, Z_0 Z'_0)\}$ ,
- ▶  $\delta'(q, \varepsilon, X) = \delta(q, \varepsilon, X) \cup \{(q_e, X)\}$  for  $q \in F$  and  $X \in \Gamma$ ,
- ▶  $\delta'(q_e, \varepsilon, X) = \{(q_e, \varepsilon)\}$  for every  $X \in \Gamma \cup \{Z'_0\}$ .

## From empty stack to accepting state

Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$  be a PDA.

The goal: Define a PDA  $M'$  such that  $L(M') = N(M)$ .

The intuition of  $M'$ :

*Add a new start symbol  $Z'_0$  to  $M$ , if  $Z'_0$  becomes the topmost symbol of the stack, then  $M'$  enters a final state.*

$M' = (Q \cup \{q'_0, q_f\}, \Sigma, \Gamma \cup \{Z'_0\}, \delta', q'_0, Z'_0, \{q_f\})$  is defined as follows.  
 $\delta'$  includes all the elements of  $\delta$  and the transitions

- ▶  $\delta'(q'_0, \varepsilon, Z'_0) = \{(q_0, Z_0 Z'_0)\},$
- ▶  $(q_f, \varepsilon) \in \delta'(q, \varepsilon, Z'_0)$  for every  $q \in Q$ .

# Summary

## Theorem

*PDA without start stack symbol using accepting state*

$\equiv$

*PDA with start stack symbol using accepting state*

$\equiv$

*PDA with start stack symbol using empty stack*

Note: PDA without start stack symbol using empty stack as accepting condition is weak than others, as its language always contains  $\varepsilon$ .

# Outline

## Context-free Grammar and Pushdown Automata

- Context-free Grammar

- Ambiguity

- Chomsky Normal Form

- Pushdown Automata

- Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

- Pumping lemma

- Closure Properties

## Deterministic Context-Free Languages

- Deterministic Context-Free Languages

- Properties of DCFLs

- Deterministic context-free grammars

- Relationship of DPDAs and DCFGs

# From CFG to PDA

## The intuition

Use the stack to store the words reached in the **leftmost** derivation.

1. Place the start symbol of the CFG as the start stack symbol of the PDA
2. Repeat the following steps forever.
  - 2.1 If the top of stack is a non-terminal  $A$ , nondeterministically select one of the rules for  $A$  and substitute  $A$  by the string on the right-hand side of the rule.
  - 2.2 If the top of stack is a terminal symbol  $a$ , read the next symbol from the input and compare it to  $a$ . If they match, repeat. If they do not match, reject on this branch of the nondeterminism.
  - 2.3 Doing so accepts the input if the stack becomes empty.

# From CFG to PDA

## The intuition

Use the stack to store the words reached in the **leftmost** derivation.

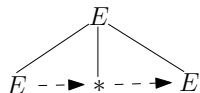
$E$



# From CFG to PDA

## The intuition

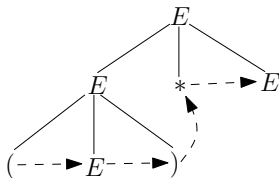
Use the stack to store the words reached in the **leftmost** derivation.



# From CFG to PDA

## The intuition

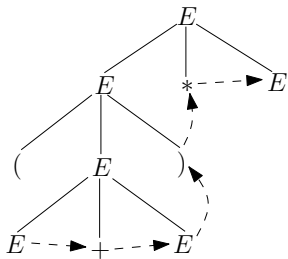
Use the stack to store the words reached in the **leftmost** derivation.



## From CFG to PDA

## The intuition

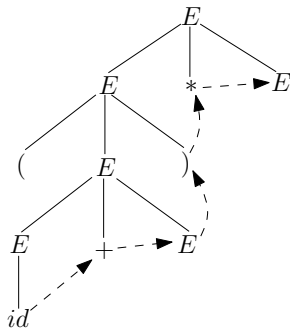
Use the stack to store the words reached in the **leftmost** derivation.



# From CFG to PDA

## The intuition

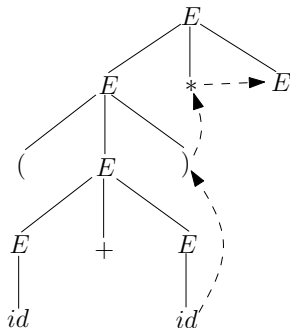
Use the stack to store the words reached in the **leftmost** derivation.



# From CFG to PDA

## The intuition

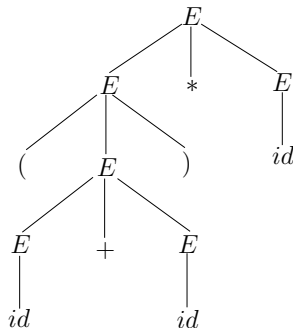
Use the stack to store the words reached in the **leftmost** derivation.



# From CFG to PDA

## The intuition

Use the stack to store the words reached in the **leftmost** derivation.



# From CFG to PDA: continued

## The construction

Let  $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$  be a CFG.

Define a PDA  $M = (\{q\}, \Sigma, \Sigma \cup \mathcal{N}, \delta, q, S, \emptyset)$  as follows.

- ▶  $\delta(q, \varepsilon, A) = \{(q, \alpha)\}$  for each production rule  $A \rightarrow \alpha$ ,
- ▶  $\delta(q, a, a) = \{(q, \varepsilon)\}$  for every  $a \in \Sigma$ .

## The correctness of the construction $N(M) = L(G)$

Claim.

$$\forall x \in \Sigma^*, \alpha \in (\mathcal{N} \cup \Sigma)^*,$$

$S \Rightarrow^* x\alpha$  by leftmost derivation in  $G$  iff  $(q, x, S) \vdash_M^* (q, \varepsilon, \alpha)$ .

## Proof of the claim.

Induction on the derivation or computation steps.

# From PDA to CFG

An illustration of the intuition:

*Analyze the computation tree of PDA  $M$   
for  $\{ww^R \mid w \in \{0,1\}^*\}$  over the input 0110.*

$$\delta(q_0, 0, Z_0) = (q_0, AZ_0)$$

$$\delta(q_0, 1, A) = (q_0, BA)$$

$$\delta(q_0, 1, B) = (q_1, \varepsilon)$$

$$\delta(q_1, 0, A) = (q_1, \varepsilon)$$

$$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$$





# From PDA to CFG

An illustration of the intuition:

*Analyze the computation tree of PDA  $M$*

*for  $\{ww^R \mid w \in \{0,1\}^*\}$  over the input 0110.*

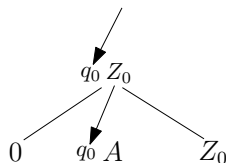
$$\delta(q_0, 0, Z_0) = (q_0, AZ_0)$$

$$\delta(q_0, 1, A) = (q_0, BA)$$

$$\delta(q_0, 1, B) = (q_1, \varepsilon)$$

$$\delta(q_1, 0, A) = (q_1, \varepsilon)$$

$$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$$



# From PDA to CFG

An illustration of the intuition:

*Analyze the computation tree of PDA  $M$*

*for  $\{ww^R \mid w \in \{0,1\}^*\}$  over the input 0110.*

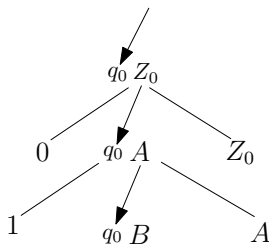
$$\delta(q_0, 0, Z_0) = (q_0, AZ_0)$$

$$\delta(q_0, 1, A) = (q_0, BA)$$

$$\delta(q_0, 1, B) = (q_1, \varepsilon)$$

$$\delta(q_1, 0, A) = (q_1, \varepsilon)$$

$$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$$



# From PDA to CFG

An illustration of the intuition:

*Analyze the computation tree of PDA  $M$*

*for  $\{ww^R \mid w \in \{0,1\}^*\}$  over the input 0110.*

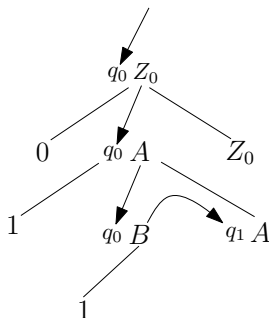
$$\delta(q_0, 0, Z_0) = (q_0, AZ_0)$$

$$\delta(q_0, 1, A) = (q_0, BA)$$

$$\delta(q_0, 1, B) = (q_1, \varepsilon)$$

$$\delta(q_1, 0, A) = (q_1, \varepsilon)$$

$$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$$



# From PDA to CFG

An illustration of the intuition:

*Analyze the computation tree of PDA  $M$*

*for  $\{ww^R \mid w \in \{0,1\}^*\}$  over the input 0110.*

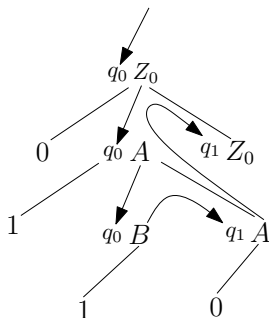
$$\delta(q_0, 0, Z_0) = (q_0, AZ_0)$$

$$\delta(q_0, 1, A) = (q_0, BA)$$

$$\delta(q_0, 1, B) = (q_1, \varepsilon)$$

$$\delta(q_1, 0, A) = (q_1, \varepsilon)$$

$$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$$



# From PDA to CFG

An illustration of the intuition:

*Analyze the computation tree of PDA  $M$*

*for  $\{ww^R \mid w \in \{0,1\}^*\}$  over the input 0110.*

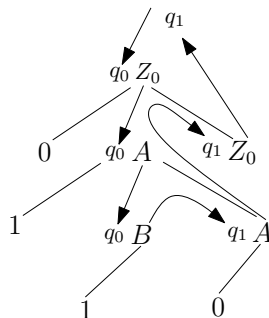
$$\delta(q_0, 0, Z_0) = (q_0, AZ_0)$$

$$\delta(q_0, 1, A) = (q_0, BA)$$

$$\delta(q_0, 1, B) = (q_1, \varepsilon)$$

$$\delta(q_1, 0, A) = (q_1, \varepsilon)$$

$$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$$



## From PDA to CFG

An illustration of the intuition:

Analyze the computation tree of PDA  $M$

for  $\{ww^R \mid w \in \{0, 1\}^*\}$  over the input 0110.

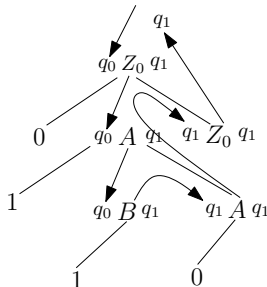
$$\delta(q_0, 0, Z_0) = (q_0, AZ_0)$$

$$\delta(q_0, 1, A) = (q_0, BA)$$

$$\delta(q_0, 1, B) = (q_1, \varepsilon)$$

$$\delta(q_1, 0, A) = (q_1, \varepsilon)$$

$$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$$



# From PDA to CFG

An illustration of the intuition:

*Analyze the computation tree of PDA  $M$*

*for  $\{ww^R \mid w \in \{0,1\}^*\}$  over the input 0110.*

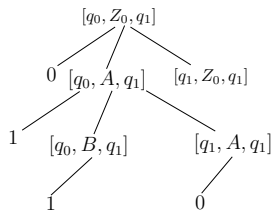
$$\delta(q_0, 0, Z_0) = (q_0, AZ_0)$$

$$\delta(q_0, 1, A) = (q_0, BA)$$

$$\delta(q_0, 1, B) = (q_1, \varepsilon)$$

$$\delta(q_1, 0, A) = (q_1, \varepsilon)$$

$$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$$



## From PDA to CFG: continued

Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$  be a PDA.

The goal: Define a CFG  $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$  such that  $L(G) = N(M)$ .

- ▶  $\mathcal{N} = Q \times \Gamma \times Q \cup \{S\}$ .
- ▶  $\mathcal{P}$  is defined as follows.
  - ▶  $S \rightarrow [q_0, Z_0, q]$  for every  $q \in Q$ .
  - ▶ If  $(p, B_1 \dots B_k) \in \delta(q, a, X)$  (where  $a \in \Sigma \cup \{\varepsilon\}$ ), then  $[q, X, q_{k+1}] \rightarrow a[q_1, B_1, q_2] \dots [q_k, B_k, q_{k+1}]$  such that  $q_1 = p$ .

### The correctness of the construction

Claim.  $[q, A, p] \Rightarrow^* x$  iff  $(q, x, A) \vdash_M^* (p, \varepsilon, \varepsilon)$ .

Proof of the claim.

Induction on the derivation or computation steps.



# Outline

## Context-free Grammar and Pushdown Automata

- Context-free Grammar

- Ambiguity

- Chomsky Normal Form

- Pushdown Automata

- Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

- Pumping lemma

- Closure Properties

## Deterministic Context-Free Languages

- Deterministic Context-Free Languages

- Properties of DCFLs

- Deterministic context-free grammars

- Relationship of DPDAs and DCFGs

# Outline

## Context-free Grammar and Pushdown Automata

- Context-free Grammar

- Ambiguity

- Chomsky Normal Form

- Pushdown Automata

- Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

- Pumping lemma

- Closure Properties

## Deterministic Context-Free Languages

- Deterministic Context-Free Languages

- Properties of DCFLs

- Deterministic context-free grammars

- Relationship of DPDAs and DCFGs

## Example

$\{a^i b^i c^i \mid i \geq 1\}$  is not context free.

How to **PROVE** it is not context free?

## Pumping lemma

**Pumping lemma.** For every CFL  $L$ , there is a natural number  $p \geq 1$  (the pumping length) such that for every string  $z \in L$  with  $|z| \geq p$ ,  $z$  can be decomposed into  $uvwxy$  satisfying that

- ▶  $|vx| \geq 1$ ,
- ▶  $|vwx| \leq p$ ,
- ▶ for every  $i \in \mathbb{N}$ ,  $uv^iwx^iy \in L$ .

Let  $L$  be a CFL and  $G = (\mathcal{N}, \Sigma, \mathcal{P}, S)$  be a CFG s.t.  $L = L(G)$ .

- ▶ Let  $k$  be the number of nonterminals in  $G$
- ▶ Let  $b$  be the maximum number of symbols in the right side of a rule

In any parse tree using this grammar, every node can have no more than  $b$  children. So, if the height of the parse tree is at most  $h$  ( $h = 0$  for root), the length of the string generated is at most  $b^h$ . Conversely, if a generated string is at least  $b^h + 1$  long, each of its parse trees must be at least  $h + 1$  high.

We choose the pumping length

$$p = b^{k+1}.$$

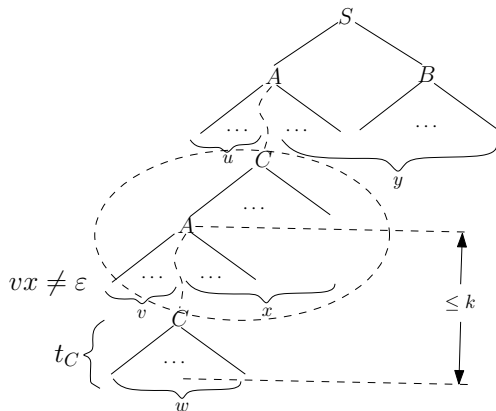
For any string  $z \in L$  with  $|z| \geq p$ , any of its parse trees must be at least  $k + 1$  high.

## Pumping lemma: continued

The idea of the proof.

*Analyzing the structure of the derivation trees of CFGs.*

At least  $k + 1$  high, then, a path from the root to a leaf of length  $k + 1$  has  $k + 2$  nodes. One non-terminal must appear at least **twice** in the last  $k + 1$  non-terminal nodes on this path.

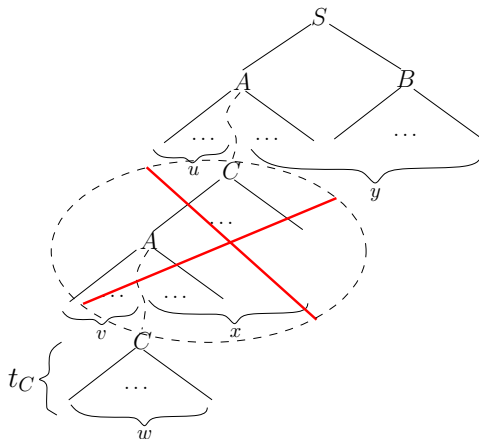


# Pumping lemma: continued

The idea of the proof.

*Analyzing the structure of the derivation trees of CFGs.*

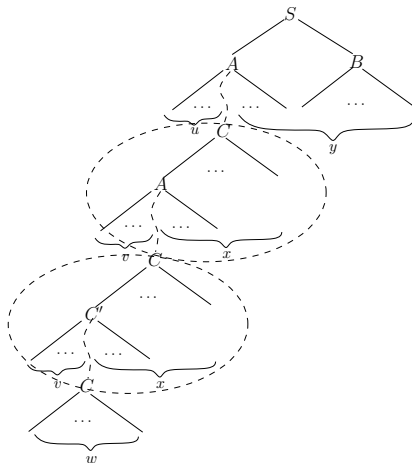
If  $vx = \varepsilon$ , then consider the reduced derivation tree instead



# Pumping lemma: continued

The idea of the proof.

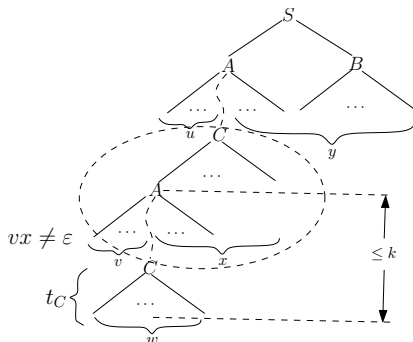
*Analyzing the structure of the derivation trees of CFGs.*



# Pumping lemma: continued

The idea of the proof.

*Analyzing the structure of the derivation trees of CFGs.*



$|vwx|$ : # leaves of a tree of height  $\leq k + 1$ :  $\leq b^{k+1}$ .

Note:  $p := b^{k+1}$ .



# Applications of pumping lemma

**Proposition.**  $L = \{a^i b^i c^i \mid i \geq 1\}$  is not a CFL.

To the contrary, suppose that  $L$  is a CFL.

Then there is a CFG  $G$  such that  $L = L(G)$ .

Let  $p \geq 1$  be the pumping length.

Consider  $z = a^p b^p c^p$ .

Then  $z = uvwxy$  s.t.  $|vx| \geq 1$  and  $|vwx| \leq p$ .  $\exists$  the following situations

- ▶  $vwx = a^i$  or  $vwx = b^i$  or  $vwx = c^i$  for some  $i : 1 \leq i \leq p$ .  
*Consequences:  $uv^2wx^2y \notin L$ , a contradiction.*
- ▶  $vwx = a^i b^j$  or  $vwx = b^i c^j$  for some  $i, j : 1 \leq i, j, i + j \leq p$ .  
*Consequences:  $uv^2wx^2y \notin L$ , a contradiction.*

# Outline

## Context-free Grammar and Pushdown Automata

- Context-free Grammar

- Ambiguity

- Chomsky Normal Form

- Pushdown Automata

- Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

- Pumping lemma

- Closure Properties

## Deterministic Context-Free Languages

- Deterministic Context-Free Languages

- Properties of DCFLs

- Deterministic context-free grammars

- Relationship of DPDAs and DCFGs

# Closure properties

**Theorem.** The class of CFLs is

- ▶ closed under union,
- ▶ **not** closed under intersection,
- ▶ closed under intersection with regular languages,
- ▶ **not** closed under complementation.

# Union

Suppose  $G_1 = (\mathcal{N}_1, \Sigma, \mathcal{P}_1, S_1)$  and  $G_2 = (\mathcal{N}_2, \Sigma, \mathcal{P}_2, S_2)$ .

Then  $G$ : Add the rules  $S \rightarrow S_1, S \rightarrow S_2$ .

It is easy to see that  $L(G) = L(G_1) \cup L(G_2)$ .

# Intersection

$L_1 = \{a^i b^j c^j \mid i, j \geq 1\}$  and  $L_2 = \{a^j b^i c^i \mid i, j \geq 1\}$  are both CFL.

On the other hand,  $L_1 \cap L_2 = \{a^i b^i c^i \mid i \geq 1\}$  is not a CFL.

# Intersection with regular languages

Let

- ▶  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  be a PDA for  $L_1$ ,
- ▶  $\mathcal{A} = (Q', \Sigma, \delta', q'_0, F')$  be a NFA for  $L_2$ .

Then  $L_1 \cap L_2 = L(M'')$  such that

$$M'' = (Q \times Q', \Sigma, \Gamma, \delta'', (q_0, q'_0), Z_0, F \times F'),$$

where

- ▶  $\delta''((q, q'), a, X) = \{((p, p'), \gamma) \mid (p, \gamma) \in \delta(q, a, X), \delta'(q', a) = p'\}$   
for  $a \in \Sigma$ ,
- ▶  $\delta''((q, q'), \varepsilon, X) = \{((p, q'), \gamma) \mid (p, \gamma) \in \delta(q, \varepsilon, X)\}.$

# Complementation

If the class of CFLs is closed under complementation, then  
from the fact that  $L_1 \cap L_2 = \overline{(\overline{L_1} \cup \overline{L_2})}$ ,  
the class of CFLs is closed under intersection, a contradiction.

# Outline

## Context-free Grammar and Pushdown Automata

- Context-free Grammar

- Ambiguity

- Chomsky Normal Form

- Pushdown Automata

- Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

- Pumping lemma

- Closure Properties

## Deterministic Context-Free Languages

- Deterministic Context-Free Languages

- Properties of DCFLs

- Deterministic context-free grammars

- Relationship of DPDAs and DCFGs



# Outline

## Context-free Grammar and Pushdown Automata

- Context-free Grammar

- Ambiguity

- Chomsky Normal Form

- Pushdown Automata

- Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

- Pumping lemma

- Closure Properties

## Deterministic Context-Free Languages

- Deterministic Context-Free Languages

- Properties of DCFLs

- Deterministic context-free grammars

- Relationship of DPDAs and DCFGs

# Deterministic pushdown automata

## Definition

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

1.  $Q$  is a finite set of states,
2.  $\Sigma$  is a finite set of input alphabet,
3.  $\Gamma$  is a finite set of stack alphabet,
4.  $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow (Q \times \Gamma_\epsilon) \cup \{\emptyset\}$  is the transition function,
5.  $q_0 \in Q$  is the start state, and
6.  $F \subseteq Q$  is the set of accept states.

For every  $q \in Q$ ,  $a \in \Sigma$ , and  $x \in \Gamma$ , exactly one of the values

$$\delta(q, a, x), \delta(q, a, \epsilon), \delta(q, \epsilon, x), \text{ and } \delta(q, \epsilon, \epsilon)$$

is not  $\emptyset$ .

**Intuition:** at most one action at any configuration

# Deterministic Context-free Language

1. The language of a DPDA is a **deterministic context-free language** (DCFL),
2. A language can be represented by a DPDA then it is **not** inherently ambiguous

Regular  $\subset$  DCFL  $\subset$  Inherently Unambiguous  $\subset$  CFL

3.  $L = \{a^n b^n c^m d \mid m, n \geq 0\} \cup \{a^n b^m c^m e \mid m, n \geq 0\}$  Inherently Unambiguous but NOT DCFL
4.  $\{0^i 1^i \mid i \geq 0\}$  is DCFL
5.  $\{a^i b^j c^k \mid i, j, k \geq 0, i = j \vee i = k\}$  is CFL but not DCFL (have to guess whether  $i = j$  or  $i = k$ )
6.  $\{ww^R \mid w \in \Gamma^*\}$  is not DCFL (have to guess when  $w^R$  starts)

DPDA may reject inputs by failing to read the entire input string, but such DPDA introduces messy cases.

### Lemma

*Every DPDA has an equivalent DPDA that always reads the entire input string.*

## Proof (1)

Let  $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$  be a DPDA.  $P$  may fail to read the entire input for two reasons:

1. It tries to pop an empty stack – **hanging**.
2. It makes an endless sequence of  $\varepsilon$ -input moves – **looping**.

To solve 1, we initialize the stack with  $\$$ . If  $\$$  is popped from the stack before the end of the input,  $P$  reads to the end of the input and rejects.

For 2, we identify the looping situations, i.e., those from which no further input symbol is ever read, and reprogramming  $P$  so that it reads and rejects the input instead of looping.

## Proof (2)

Add new states:

$q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}}$ .

(i) Once  $P$  enters an accept state, it remains in accepting states until it reads the next input symbol:

- ▶ Add a new accept state  $q_a$  for every  $q \in Q$ .
- ▶ For every  $q \in Q$ , if  $\delta(q, \varepsilon, x) = (r, y)$ , then set  $\delta(q_a, \varepsilon, x) = (r_a, y)$ .  
If  $q \in F$ , also change  $\delta(q, \varepsilon, x) = (r_a, y)$ .
- ▶ For each  $q \in Q$  and  $a \in \Sigma$ , if  $\delta(q, a, x) = (r, y)$ , then  $\delta(q_a, a, x) = (r, y)$ .
- ▶ Let  $F'$  be the set of new and old accept states.

## Proof (3)

(ii)  $P$  rejects if it tries to pop an empty stack.

- ▶  $P$  initializes the stack with the symbol  $\$$  by  $\delta(q_{\text{start}}, \varepsilon, \varepsilon) = (q_0, \$)$ .
- ▶ If  $P$  detects  $\$$  while in a non-accepting state, it enters  $q_{\text{rejects}}$  and scans the input to the end.
- ▶ If  $P$  detects  $\$$  while in an accept state, it enters  $q_{\text{accept}}$ . Then if any input remains unread, it enters  $q_{\text{reject}}$  and scans the input to the end.
- ▶ More precisely, for  $x \in \Gamma$  and  $\delta(q, a, x) \neq \emptyset$ ,
  - ▶ if  $q \notin F'$  and  $a \in \Sigma$ , then set  $\delta(q, a, \$) = (q_{\text{reject}}, \varepsilon)$ .
  - ▶ if  $q \in F'$  and  $a = \varepsilon$ , then set  $\delta(q, \varepsilon, \$) = (q_{\text{accept}}, \varepsilon)$ .
  - ▶ for  $a \in \Sigma$ , set  $\delta(q_{\text{reject}}, a, \varepsilon) = (q_{\text{reject}}, \varepsilon)$  and  $\delta(q_{\text{accept}}, a, \varepsilon) = (q_{\text{reject}}, \varepsilon)$ .

## Proof (4)

(iii) Modify  $P$  to reject instead of making an endless sequence of  $\varepsilon$ -input moves.

- ▶ For every  $q \in Q$  and  $x \in \Gamma$ , call  $(q, x)$  a **looping situation** if, when  $P$  is started in state  $q$  with  $x \in \Gamma$  on the top of the stack, if it never pops anything below  $x$  and it never reads an input symbol.
- ▶ A loop situation is **accepting**, if  $P$  enters an accept state during its subsequent moves, and otherwise **rejecting**.
- ▶ If  $(q, x)$  is an accepting looping situation, set  $\delta(q, \varepsilon, x) = (q_{\text{accept}}, \varepsilon)$ .
- ▶ If  $(q, x)$  is a rejecting looping situation, set  $\delta(q, \varepsilon, x) = (q_{\text{reject}}, \varepsilon)$ .

Note that: for  $a \in \Sigma$ , set  $\delta(q_{\text{reject}}, a, \varepsilon) = (q_{\text{reject}}, \varepsilon)$  and  $\delta(q_{\text{accept}}, a, \varepsilon) = (q_{\text{reject}}, \varepsilon)$ .



# Outline

## Context-free Grammar and Pushdown Automata

- Context-free Grammar

- Ambiguity

- Chomsky Normal Form

- Pushdown Automata

- Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

- Pumping lemma

- Closure Properties

## Deterministic Context-Free Languages

- Deterministic Context-Free Languages

- Properties of DCFLs

- Deterministic context-free grammars

- Relationship of DPDAs and DCFGs

## Theorem

*The class of DCFLs is not closed under union and intersection.*

- ▶  $L_1 = \{a^i b^j c^k \mid i, j, k \geq 0, i = j\}$  is DCFL
- ▶  $L_2 = \{a^i b^j c^k \mid i, j, k \geq 0, i = k\}$  is DCFL
- ▶  $L_1 \cup L_2 = \{a^i b^j c^k \mid i, j, k \geq 0, i = j \vee i = k\}$  is CFL but not DCFL (have to guess whether  $i = j$  or  $i = k$ )
- ▶  $L_1 \cap L_2 = \{a^i b^j c^k \mid i, j, k \geq 0, i = j = k\}$  is not CFL hence not DCFL

## Theorem

*The class of DCFLs is closed under complementation. That is, if  $A$  is a DCFL, then*

$$\Sigma^* - A = \{s \in \Sigma^* \mid s \notin A\}$$

*is also a DCFL.*

## Proof (1)

We cannot simply **swap the accept and non-accept states** of a DPDA. The DPDA may accept its input by entering both accept and non-accept states in a sequence of moves at the end of the input string.

For example,  $(q_0, w, \varepsilon) \rightarrow^* (q, \varepsilon, \gamma) \rightarrow^+ (q', \varepsilon, \gamma')$ ,  $q \notin F$ , but  $q' \in F$

Assume  $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$  is a DPDA for  $A$  which always reads the whole input string. Moreover, once  $P$  enters an accept state, it remains in accept states until it reads the next input symbol.

## Proof (2)

We designate some states as **reading** states which never pop symbols from the stack

- ▶ If  $P$  in a state  $q$  reads an  $a \in \Sigma$  without popping the stack, i.e.,  $\delta(q, a, \varepsilon) \neq \emptyset$ , then  $q$  is a reading state.
- ▶ If  $\delta(q, a, x) = (r, y)$ , then add a new state  $q_x$  and modify  $\delta$  as

$$\delta(q, \varepsilon, x) = (q_x, \varepsilon) \quad \text{and} \quad \delta(q_x, a, \varepsilon) = (r, y).$$

Let  $q_x$  be a reading state, and it is an accept state if  $q \in F$ .

Remove the accepting state designation from any state which isn't a reading state, namely, only reading states can be accepting states

## Proof (3)

Now, invert which reading states are classified as accepting. The resulting DPDA recognizes the complementary language.

## Corollary

*Any CFL whose complement is not a CFL is not a DCFL.*

Intuition: Consider the CFL  $L$  such that  $\bar{L}$  is not a CFL. Assume  $L$  is DCFL, then,  $\bar{L}$  is a DCFL, hence CFL, contradiction.

## Example

$\{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$  is a CFL but not a DCFL.

# Endmarked languages

## Definition

For any language  $A$  the **endmarked language**  $A\vdash$  is defined by

$$\{w\vdash \mid w \in A\}.$$

Here  $\vdash$  is the special endmarker symbol denoting the end of input.

## Theorem

*$A$  is a DCFL if and only if  $A\vdash$  is a DCFL.*



## Proof (1)

Let  $P$  be an DPDA recognizing  $A$ . Then DPDA  $P'$  recognizes  $A\vdash$  by:

1. Simulating  $P$  until  $P'$  reads  $\vdash$ .
2.  $P'$  accepts if  $P$  had entered an accept state during the previous symbol.
3.  $P'$  does not read any symbols after  $\vdash$ .

Assume that only reading states in  $P$  can be accepting states and once enters an accepting state, it remains in accepting states until it reads the next input symbol.

For every  $q \in F$ ,  $\delta'(q, \vdash, x) = (q_f, \varepsilon)$  for all  $x \in \Gamma_\varepsilon$  and  $q_f$  is the unique accepting state.

Why do we need the above assumption? To be deterministic, reading states  $q$  gives  $\delta(q, a, x) \neq \emptyset$ , then  $\delta(q, \varepsilon, x) = \emptyset$

## Proof (2)

Let DPDA  $P = (Q, \Sigma \cup \{\neg\}, \Gamma, \delta, q_0, F)$  recognize  $A\neg$ . Construct a DPDA  $P'$  recognizing  $A$ .

Intuition:

- ▶  $P'$  reads input and simulates  $P$
- ▶ Before reading the next input symbol,  $P'$  determines whether  $P$  would accept if that input symbol is  $\neg$
- ▶ If yes, then  $P'$  enters an accepting state
- ▶ Note:  $P$  may eventually enter an accepting state after reading  $\neg$  by manipulating the stack
- ▶ How to predicate future states are accepting or not?
- ▶ We will store additional information ( $R$ ) on the stack of  $P'$  which allows  $P'$  to determine immediately whether  $P$  would accept
- ▶  $R$  contains states of  $P$  from which  $P$  eventually enters an accepting state but without reading further input

## Proof (3)

Let DPDA  $P = (Q, \Sigma \cup \{\vdash\}, \Gamma, \delta, q_0, F)$  recognize  $A \vdash$ .

Modify  $P$  so that each of its moves does exactly one of the following operations:

- ▶ read an input symbol, i.e.,  $\delta(q, a, \varepsilon) = (q', \varepsilon)$ ;
- ▶ push a symbol onto the stack, i.e.,  $\delta(q, \varepsilon, \varepsilon) = (q', \gamma)$ ;
- ▶ or pop a symbol from the stack, i.e.,  $\delta(q, \varepsilon, x) = (q', \varepsilon)$ .

We construct a  $P'$  to simulate  $P$ . Every time  $P'$  pushes a stack symbol of  $P'$ , then it pushes a symbol representing a subset of  $P$ 's states, i.e.,  $\Gamma' = \Gamma \cup \mathcal{P}(Q)$ . Then, the stack content of  $P'$  would be  $R_{k+1}\gamma_k \cdots R_2\gamma_1 R_1\gamma_0 R_0$ , by starting  $P$  in any state of  $R_{k+1}$  with stack content  $\gamma_k \cdots \gamma_1\gamma_0$  will eventually accept without reading any more input.

Initially,  $P'$  pushes the set  $R_0$  defined by

$$R_0 = \{q \in Q \mid (q, \varepsilon, \varepsilon) \vdash_M^* (q', \varepsilon, u) \text{ for some } q' \in F, u \in \Gamma^*\}$$

when  $P$  is started in  $q$  with an empty stack, it eventually accepts without reading any input symbols

## Proof (4)

Then  $P'$  begins simulating  $P$ .

- ▶ To simulate a pop move,  $P'$  first pops and discards the set of states on the top of the stack, then it pops again to obtain the symbol in order to simulate  $P$ , i.e., if  $\delta(q, \varepsilon, x) = (q', \varepsilon)$ , then  $\delta'(q, \varepsilon, R) = (q_x, \varepsilon)$  and  $\delta'(q_x, \varepsilon, x) = (q', \varepsilon)$
- ▶ To simulate a push move  $\delta(q, \varepsilon, \varepsilon) = (q', x)$ ,  $P'$  examines the set of states  $R$  on the top of its stack, and then it pushes  $x$  and the set of states

$$\{q'' \mid q'' \in F \text{ or } (\delta(q'', \varepsilon, x) = (r, \varepsilon) \text{ and } r \in R)\}.$$

- ▶  $P'$  simulates a read move  $\delta(q, a, \varepsilon) = (q', \varepsilon)$ , by examining the set  $R$  on the top of the stack and entering an accept state if  $q' \in R$ .
  - ▶ If  $P'$  is at the end of the input string, i.e.,  $a = \perp$ , then it accepts.
  - ▶ Otherwise i.e.,  $a \neq \perp$ , it will continue simulating  $P$ , so this accept state must also record  $P$ 's state.

# Outline

## Context-free Grammar and Pushdown Automata

- Context-free Grammar

- Ambiguity

- Chomsky Normal Form

- Pushdown Automata

- Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

- Pumping lemma

- Closure Properties

## Deterministic Context-Free Languages

- Deterministic Context-Free Languages

- Properties of DCFLs

- Deterministic context-free grammars

- Relationship of DPDAs and DCFGs

Our goal is to define **deterministic context-free grammars (DCFG)**, the counterpart to deterministic pushdown automata.

We will show that these two models are equivalent on **endmarked languages**.

# From derivations to reductions

Derivations in CFGs begin with the start non-terminal and proceed **top down** with a series of substitutions according to the grammar's rules, until the derivation obtains a string of terminals.

For defining DCFGs we take a **bottom up** approach, by starting with a string of terminals and processing the derivation in reverse, employing a series of reduce steps until reaching the start variable.

- ▶ Each **reduce step** is a reversed substitution, whereby the string of terminals and non-terminals on the right-hand side of a rule is replaced by the non-terminal on the corresponding left-hand side.
- ▶ The string replaced is the **reducing string**.
- ▶ The entire reversed derivation is a **reduction**.

# Formal definition of reductions

If  $u$  and  $v$  are strings of non-terminals and terminals, write  $u \rightarrow v$  to mean that  $v$  can be obtained from  $u$  by a reduce step, i.e.,  $v \Rightarrow u$ .

A reduction from  $u$  to  $v$  is a sequence

$$u = u_1 \rightarrow u_2 \rightarrow \dots \rightarrow u_k = v$$

and we say that  $u$  is reducible to  $v$ , written  $u \xrightarrow{*} v$ , equivalent  $v \xRightarrow{*} u$ .

A reduction from  $u$  is a reduction from  $u$  to the start non-terminal.

In a leftmost reduction, each reducing string is reduced only after all other reducing strings that lie entirely to its left.

This is the reverse of rightmost derivation



Let  $w \in L(G)$ , and let  $u_i$  appear in a leftmost reduction of  $w$ . In the reduce step  $u_i \rightarrow u_{i+1}$ , say that the rule  $T \rightarrow h$  was applied in reverse.

Therefore,

$$u_i = xhy \quad \text{and} \quad u_{i+1} = xTy,$$

where  $h$  is the reducing string.

$(h, T \rightarrow h)$  is a **handle** of  $u_i$ . A string that appears in a leftmost reduction of some string in  $L(G)$  is called a **valid string**. We define handles only for valid strings.

Valid string may have several handles, but only if **the grammar is ambiguous**.

Unambiguous grammars generate strings by one parse tree only, and therefore the leftmost reductions, and hence the handles, are also unique. In that case, we may refer to **the handle** of a valid string.

In

$$u_i = xhy \quad \text{and} \quad u_{i+1} = xTy,$$

$y$  is always a string of terminals.

## Example (1)

Consider the grammar  $G$ :

$$R \rightarrow S \mid T$$

$$S \rightarrow aSb \mid ab$$

$$T \rightarrow aTbb \mid abb$$

Then

$$L(G) = B \cup C$$

$$\text{where } B = \{a^m b^m \mid m \geq 1\}$$

$$\text{and } C = \{a^m b^{2m} \mid m \geq 1\}.$$

Some leftmost reductions:

$$aa\underline{ab}bb \rightarrow aa\underline{S}bb \rightarrow \underline{aSb} \rightarrow \underline{S} \rightarrow R,$$

$$aa\underline{ab}bbbb \rightarrow aa\underline{T}bbbb \rightarrow \underline{aTbb} \rightarrow \underline{T} \rightarrow R.$$

## Example (2)

Consider the grammar  $G$ :

$$S \rightarrow T \dashv$$

$$T \rightarrow T(T) \mid \varepsilon$$

A leftmost reduction:

$$\_()() \dashv \rightsquigarrow T(\_>() \dashv \rightsquigarrow \underline{T(T)}() \dashv \rightsquigarrow T(\_) \dashv \rightsquigarrow \underline{T(T)} \dashv \rightsquigarrow \underline{T} \dashv \rightsquigarrow S.$$

## Definition

A handle  $h$  of a valid string  $v = xhy$  is a **forced handle** if  $h$  is **the unique handle in every valid string  $xh\hat{y}$  where  $\hat{y} \in \Sigma^*$** .

## Definition

A **deterministic context-free grammar (DCFG)** is a context-free grammar such that every valid string has a forced handle.

The above definition does not tell us how to decide whether a given CFG is a DCFG.

## DK-test (Donald Knuth)

A CFG  $G$  is DCFG iff  $G$  passes  $DK$ -test.

For any CFG we construct an associated DFA  $DK$  that can identify handles.  $DK$  accepts its input  $z$  if

1.  $z$  is the prefix of some valid string  $v = zy$ , and
2.  $z$  ends with a handle of  $v$ .

Moreover, each accept state of  $DK$  indicates the associated reducing rule(s). In a general CFG, multiple reducing rules may apply, depending on which valid  $v$  extends  $z$ . But in a DCFG, **each accept state corresponds to exactly one reducing rule.**

## Two provisos

1. The start non-terminal of a CFG does not appear on the right-hand side of any rule.
2. Every non-terminal appears in a reduction of some string in the grammar's language, i.e., no useless production rules.

# The plan

To construct DFA  $DK$ , we will construct an equivalent NFA  $K$  and convert  $K$  to  $DK$  via the subset construction.

To understand  $K$ , we introduce an NFA  $J$  which  
*accepts every input string that ends with the right-hand side of any rule.*



# The NFA $J$

1. guesses a rule  $B \rightarrow u$  to use,
2. guesses at which point to start matching the input with  $u$ ,
3. keeps track of its progress through  $u$ .

We represent this progress by placing a **dot** in the corresponding point in the rule, yielding the following **dotted rules**:

$$B \rightarrow .u_1u_2 \cdots u_{k-1}u_k$$

$$B \rightarrow u_1.u_2 \cdots u_{k-1}u_k$$

$$\vdots$$

$$B \rightarrow u_1u_2 \cdots u_{k-1}.u_k$$

$$B \rightarrow u_1u_2 \cdots u_{k-1}u_k.$$

## Dotted rules

1. Each dotted rule corresponds to one state of  $J$ , i.e., for  $B \rightarrow u.v$  we have a state  $B \rightarrow u.v$ .
2. The accept states  $B \rightarrow u.$  correspond to the complete rules.
3. We add a separate start state with a self-loop on all symbols and an  $\varepsilon$ -move to  $B \rightarrow .u$  for each rule  $B \rightarrow u$ .

Thus,  $J$  accepts if the match completes successfully at the end of the input. If a mismatch occurs or if the end of the match does not coincide with the end of the input, this branch of  $J$ 's computation rejects.

# The NFA $K$

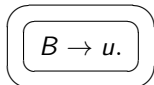
- ▶ Like  $J$ , the states of  $K$  correspond to all dotted rules.
- ▶ It has a special start state that has an  $\varepsilon$ -move to  $S_1 \rightarrow .u$  for every rule with  $S_1$  being the start variable.
- ▶ **Shift-moves:** for a rule  $B \rightarrow uav$  where  $a$  can be a terminal or non-terminal

$$B \rightarrow u.av \xrightarrow{a} B \rightarrow ua.v$$

- ▶  **$\varepsilon$ -moves:** for a rule  $B \rightarrow uCv$  and  $C \rightarrow r$

$$B \rightarrow u.Cv \xrightarrow{\varepsilon} C \rightarrow .r$$

- ▶ The accept states are all  $B \rightarrow u.$ .



## Lemma

$K$  *may* enter state  $T \rightarrow u.v$  on reading input  $z$  if and only if  $z = xu$  and  $xuvy$  is a valid string with handle  $uv$  and reducing rule  $T \rightarrow uv$ , for some  $y \in \Sigma^*$ .

# Proof (1)

First step is to compute  $xu$

Assume  $K$  enters state  $T \rightarrow u.v$  on reading input  $z$ , whose path from the start state is viewed as runs of shift-moves separated by  $\epsilon$ -moves.

- ▶ The shift-moves are transitions between states sharing the same rule, shifting the dot rightward over symbols read from the input.
- ▶ In the  $i$ th run ( $\epsilon$ -move), the rule is  $S_i \rightarrow u_i S_{i+1} v_i$ , where  $S_{i+1}$  is the non-terminal expanded in the next run.
- ▶ The penultimate run is for rule  $S_\ell \rightarrow u_\ell T v_\ell$ , and the final run has rule  $T \rightarrow uv$ .

$$\begin{aligned} S_1 &\xrightarrow{\epsilon} .u_1 S_2 v_1 \xrightarrow{u_1}^* u_1 .S_2 v_1 \xrightarrow{\epsilon} .u_2 S_3 v_2 \xrightarrow{u_2}^* u_2 .S_3 v_2 \rightarrow^* \dots \\ &\xrightarrow{u_{\ell-1}}^* u_{\ell-1} .S_\ell v_{\ell-1} \xrightarrow{\epsilon} .u_\ell T v_\ell \xrightarrow{u_\ell}^* u_\ell .T v_\ell \xrightarrow{\epsilon} .uv \xrightarrow{u}^* uv \end{aligned}$$

- ▶ So the input  $z = u_1 u_2 \dots u_\ell u = xu$ , because the strings  $u_i$  and  $u$  were the shift-move symbols read from the input.

## Proof (2)

Second step is to compute a valid string  $xuvy$ .

$$\begin{aligned} S_1 &\xrightarrow{\varepsilon} .u_1 S_2 v_1 \xrightarrow{u_1}^* u_1 .S_2 v_1 \xrightarrow{\varepsilon} .u_2 S_3 v_2 \xrightarrow{u_2}^* u_2 .S_3 v_2 \rightarrow^* \dots \\ &\xrightarrow{u_{\ell-1}}^* u_{\ell-1} .S_{\ell} v_{\ell-1} \xrightarrow{\varepsilon} .u_{\ell} T v_{\ell} \xrightarrow{u_{\ell}}^* u_{\ell} .T v_{\ell} \xrightarrow{\varepsilon} .uv \xrightarrow{u}^* u.v \end{aligned}$$

Let  $y' = v_{\ell} \dots v_2 v_1$ , then  $xuvy'$  ( $= u_1 u_2 \dots u_{\ell} uv v_{\ell} \dots v_2 v_1$ ) is derivable in  $G$ .

- ▶ Fully expand all non-terminals that appear in  $y'$  until each non-terminal derives some string of terminals, and let  $y$  be the resulting string.
- ▶ The string  $xuvy$  is valid because it occurs in a leftmost reduction of  $w \in L(G)$  of terminals obtained by expanding all non-terminals in  $xuvy'$ .
- ▶  $uv$  is the handle in the reduction and its reducing rule is  $T \rightarrow uv$ .

This proves the direction from left to right.

## Proof (3)

Assume that string  $xuv$  is a valid string with handle  $uv$  and reducing rule  $T \rightarrow uv$ . We need to show that  $K$  may enter  $T \rightarrow u.v$  on reading input  $xu$ .

1. The parse tree is rooted at the start variable  $S_1$  and it must contain the variable  $T$ , as  $T \rightarrow uv$  is the first reduce step in the reduction of  $xuv$ .
2. Let  $S_2, \dots, S_\ell$  be the non-terminals on the path from  $S_1$  to  $T$ . All non-terminals in the parse tree that appear leftward of this path must be unexpanded, or else  $uv$  would not be the handle.
3. Each  $S_i$  leads to  $S_{i+1}$  by some rule  $S_i \rightarrow u_i S_{i+1} v_i$ :

$$S_1 \rightarrow u_1 S_2 u_k$$

$$S_2 \rightarrow u_2 S_3 u_2$$

$$\vdots$$

$$S_\ell \rightarrow u_\ell T v_\ell$$

$$T \rightarrow uv.$$

## Proof (4)

On input  $z = xu$ , the path from  $K$ 's start state to  $T \rightarrow u.v$  is:

1.  $K$  makes an  $\varepsilon$ -move to  $S_1 \rightarrow .u_1 S_2 v_1$ .
2. Reading the symbols of  $u_1$ , it performs the corresponding shift-moves until it enters  $S_1 \rightarrow u_1.S_2 v_1$  at the end of  $u_1$ .
3. It makes an  $\varepsilon$ -move to  $S_2 \rightarrow .u_2 S_3 v_2$  and continues with shift-moves on reading  $u_2$  until it reaches  $S_2 \rightarrow u_2.S_3 v_2$ . . . . .
4. After reading  $u_\ell$  it enters  $S_\ell \rightarrow u_\ell.T v_\ell$  which leads by an  $\varepsilon$ -move to  $T \rightarrow .uv$ .
5. Finally after reading  $u$  it is in  $T \rightarrow u.v$ .



## Lemma

$K$  *may* enter state  $T \rightarrow u.v$  on reading input  $z$  if and only if  $z = xu$  and  $xuvy$  is a valid string with handle  $uv$  and reducing rule  $T \rightarrow uv$ , for some  $y \in \Sigma^*$ .

## Corollary

$K$  may enter state  $T \rightarrow h.$  on reading input  $z$  if and only if  $z = xh$  and  $h$  is a handle of some valid string  $xhy$  with reducing rule  $T \rightarrow h$ .

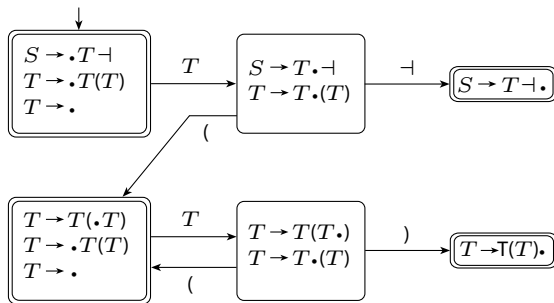
## From $K$ to $DK$

We convert NFA  $K$  to DFA  $DK$  by **the subset construction**. All states unreachable from the start state are removed.

Thus, each of  $DK$ 's states thus contains one of more dotted rules. Each accept state contains at least one completed rule.

# An example (LR(0) automaton in CS131)

$S \rightarrow T \mid$   
 $T \rightarrow T(T) \mid \epsilon$



# The $DK$ -test

Starting with a CFG  $G$ , construct the associated DFA  $DK$ . Determine whether  $G$  is deterministic by examining  $DK$ 's accept states.  $G$  passes the  $DK$ -test if every accept state contains

1. exactly one completed rule, and
2. no dotted rule in which a terminal symbol immediately follows the dot, i.e., no dotted rule of the form  $B \rightarrow u.av$  for  $a \in \Sigma$ .

## Theorem

*G passes the DK-test if and only if G is a DCFG.*

# Proof (1)

Assume that  $G$  is not deterministic and show that it fails the  $DK$ -test.

- ▶ Take a valid string  $xhy$  that has an **unforced handle**  $h$ .
- ▶ Some valid string  $xhy'$  has a different handle  $\hat{h} \neq h$ , where  $y'$  is a string of terminals, i.e.,  $y' \in \Sigma^*$ . Thus  $xhy' = \hat{x}\hat{h}\hat{y}$ .
- ▶ If  $xh = \hat{x}\hat{h}$ , then input  $xh$  sends  $DK$  to a state with two completed rules ( $T \rightarrow h$ . and  $\hat{T} \rightarrow \hat{h}$ .), failing the  $DK$ -test.
- ▶ If  $xh \neq \hat{x}\hat{h}$ , by symmetry, we assume that  $xh$  is a proper prefix of  $\hat{x}\hat{h}$ . Then  $y' \in \Sigma^+$ .
  - ▶ Let  $q$  be the state that  $DK$  enters on input  $xh$ , which must be accepting as  $h$  is a handle of  $xhy$ .
  - ▶ A transition arrow must exit  $q$  as  $\hat{x}\hat{h}$  sends  $DK$  to an accept state via  $q$ . That transition arrow is labeled with a terminal symbol since  $y' \in \Sigma^+$ .

Hence  $q$  contains a dotted rule with a terminal symbol immediately following the dot, violating the  $DK$ -test.

## Proof (2)

Assume  $G$  fails the  $DK$ -test at some accept state  $q$ . We show that  $G$  has an **unforced handle**.

$q$  has a complete rule  $T \rightarrow h.$ , for it is accepting. Let  $z$  be a string that leads  $DK$  to  $q$ . Then  $z = xh$  where some valid string  $xhy$  has handle  $h$  with reducing rule  $T \rightarrow h$ , for  $y \in \Sigma^*$ .

- ▶ If  $q$  has another completed rule  $B \rightarrow \hat{h}.$ . Then some valid string  $xhy'$  must have a different handle  $\hat{h}$  with reducing rule  $B \rightarrow \hat{h}$ . Hence,  $h$  is not a forced handle.
- ▶  $q$  contains a rule  $B \rightarrow u.av$  with  $a \in \Sigma$ .

## Proof (3)

$q$  has a complete rule  $T \rightarrow h$ ., for it is accepting. Let  $z$  be a string that leads  $DK$  to  $q$ . Then  $z = xh$  where some valid string  $xhy$  has handle  $h$  with reducing rule  $T \rightarrow h$ , for  $y \in \Sigma^*$ .

- ▶  $q$  contains a rule  $B \rightarrow u.av$  with  $a \in \Sigma$ .
  - ▶  $xh$  takes  $DK$  to  $q$ , so  $xh = \hat{x}u$ , where  $\hat{x}uav\hat{y}$  is a valid string and has a handle  $uav$  with reducing rule  $B \rightarrow uav$ , for some  $\hat{y} \in \Sigma^*$ .
  - ▶ Expand all non-terminals in  $v$  to get  $v' \in \Sigma^*$ , and let  $y' = av'\hat{y}$  with  $y' \in \Sigma^+$ .
  - ▶ The following is a leftmost reduction

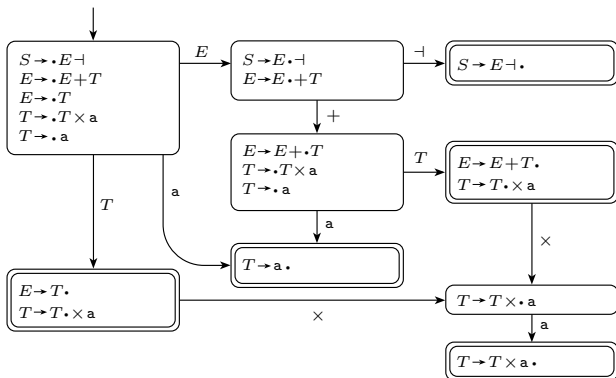
$$xhy' = xhav'\hat{y} = \hat{x}uav'\hat{y} \xrightarrow{*} \hat{x}uav\hat{y} \xrightarrow{*} \hat{x}B\hat{y} \xrightarrow{*} S.$$

- ▶  $\hat{x}uav\hat{y}$  is **valid**, and we can obtain  $\hat{x}uav'\hat{y}$  from it using a **rightmost derivation**, so  $\hat{x}uav'\hat{y}$  is also **valid**.
- ▶ The handle of  $\hat{x}uav'\hat{y}$  either lies inside  $v'$  (if  $v \neq v'$ ) or is  $uav$  (if  $v = v'$ ). In either case, the handle includes  $a$  or follows  $a$  and thus cannot be  $h$  because  $h$  fully precedes  $a$ . Hence  $h$  is a **unforced handle**.



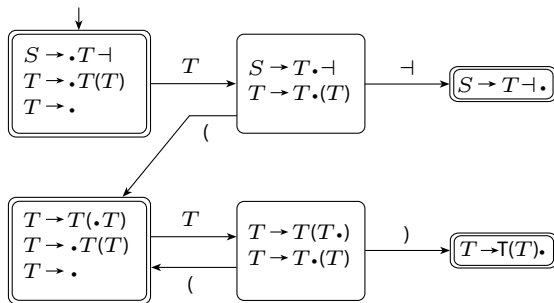
## An example failing the *DK*-test

$$\begin{aligned} S &\rightarrow E \neg \\ E &\rightarrow E + T \mid T \\ T &\rightarrow T \times a \mid a \end{aligned}$$



## An example passing the *DK*-test

$$\begin{aligned} S &\rightarrow T \dashv \\ T &\rightarrow T(T) \mid \varepsilon \end{aligned}$$



# Outline

## Context-free Grammar and Pushdown Automata

- Context-free Grammar

- Ambiguity

- Chomsky Normal Form

- Pushdown Automata

- Equivalence of Context-free Grammar and Pushdown Automata

## Properties of Context-free Languages

- Pumping lemma

- Closure Properties

## Deterministic Context-Free Languages

- Deterministic Context-Free Languages

- Properties of DCFLs

- Deterministic context-free grammars

- Relationship of DPDAs and DCFGs

# Relationship of DPDAs and DCFGs

- ▶ For every DCFG  $G$  of any context-free language, we can construct a DPDA  $P$  such that  $L(P) = L(G)$
- ▶ For every DPDA  $P$  of any endmarked language, we can construct a DCFG  $G$  such that  $L(P) = L(G)$
- ▶ Endmarkers do not affect the class of languages that DPDAs recognize, but they do affect the class of languages of DCFG
- ▶ Without endmarkers, DCFG **only** generated a strict subclass (prefix-free) of DCFL

## Theorem

*An **endmarked** language is generated by a deterministic context-free grammar if and only if it is deterministic context free.*

# Recall Endmarked languages

## Definition

For any language  $A$  the **endmarked language**  $A\vdash$  is defined by

$$\{w\vdash \mid w \in A\}.$$

Here  $\vdash$  is the special endmarker symbol.

## Theorem

*$A$  is a DCFL if and only if  $A\vdash$  is a DCFL.*

# Prefix-free languages

## Definition

$A \subseteq \Sigma^*$  is **prefix-free** if for every  $w \in A$  and every proper prefix  $w'$  of  $w$  we have  $w' \notin A$ .

## Lemma

*Every  $A^+$  is prefix-free.*

## Lemma

*Every DCFG generates a prefix-free language.*

## Prefix-free languages (cont'd)

The grammar

$$S \rightarrow T\vdash$$

$$T \rightarrow T(T) \mid \varepsilon$$

is DCFG, which recognizes some  $A\vdash$

$A$  **cannot** be generated by DCFG, since it is **not** prefix-free.

## Lemma

*Every DCFG has an equivalent DPDA.*

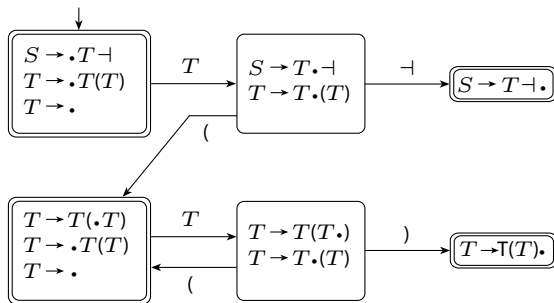
Intuition:

- ▶ Construct the DFA  $DK$  for a DCFG  $G$
- ▶ The DPDA  $P$  simulates the run of  $DK$  on an input string
  - ▶  $DK$  reads an input symbol  $a$  and moves from state  $q_1$  to  $q_2$  iff  $P$  reads  $a$  when the top of the stack is  $q_1$  and pushes  $a$  and  $q_2$  into the stack
  - ▶ When  $DK$  reaches an accepting state  $q$ , it necessarily contains a unique dotted rule  $T \rightarrow u.$  and the projection of top- $2|u|$  symbols in the stack to the terminals and non-terminals is  $u$ ,
  - ▶ pops the top- $2|u|$  symbols from the stack,
  - ▶ suppose now  $q_1$  is the state in the top of the stack,  $DK$  moves from  $q_1$  to  $q_2$  after reading  $T$ , then pushes  $T$  and  $q_2$  into the stack
  - ▶ continues the above steps until the start non-terminal  $S$  is pushed onto the stack, then  $P$  enters the accepting state



# DCFG to DPDA

$$\begin{aligned} S &\rightarrow T \dashv \\ T &\rightarrow T(T) \mid \epsilon \end{aligned}$$



## Lemma

*Every DPDA that recognizes an endmarked language has an equivalent DCFG.*

## Proof (1)

Let  $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$  be a DPDA. For each pair of states  $p$  and  $q$ , let  $A_{pq}$  be a non-terminal which generates all strings taking  $P$  from  $p$  with an empty stack to  $q$  with an empty stack.

We modify  $P$  such that:

1. It has a single accept state  $q_{\text{accept}}$ .
2. It empties its stack before accepting, which cannot be deterministic without reading the endmark  $\neg$ .
3. Each transition either pushes a symbol onto the stack or pops one off the stack, but it does not do both at the same time.

## Proof (2)

1. For every  $p, q, r, s \in Q$ ,  $u \in \Gamma$ , and  $a, b \in \Sigma_\varepsilon$ , if  $\delta(p, a, \varepsilon) = (r, u)$  and  $\delta(s, b, u) = (q, \varepsilon)$ , then add  $A_{pq} \rightarrow aA_{rs}b$ .
2. For every  $p, q, r \in Q$ , add  $A_{pq} \rightarrow A_{pr}A_{rq}$ .
3. For every  $p \in Q$ , add  $A_{pp} \rightarrow \varepsilon$ .

To avoid ambiguity, we combine rules of types 1 and 2 into:

- 1-2. For every  $p, q, r, s, t \in Q$ ,  $u \in \Gamma$ , and  $a, b \in \Sigma_\varepsilon$ , if  $\delta(r, a, \varepsilon) = (s, u)$  and  $\delta(t, b, u) = (q, \varepsilon)$ , add the rule  $A_{pq} \rightarrow A_{pr}aA_{st}b$ .

Let  $G$  be the resulting grammar.

## Proof (3)

In a derivation of the original grammar, for each substitution due to a type 2 rule  $A_{pq} \rightarrow A_{pr}A_{rq}$ , we can assume that

*$r$  is  $P$ 's state when it is at the rightmost point where the stack becomes empty midway.*

Then the subsequent substitution of  $A_{rq}$  must expand it using a type 1 rule  $A_{rq} \rightarrow aA_{st}b$ .

We can combine these two substitutions into a single type 1-2 rule

$$A_{pq} \rightarrow A_{pr}aA_{st}b.$$

In a derivation using the modified grammar  $G$ , if we replace each type 1-2 rule  $A_{pq} \rightarrow A_{pr}aA_{st}b$  by the type 2 rule  $A_{pq} \rightarrow A_{pr}A_{rq}$  followed by the type 1 rule  $A_{rq} \rightarrow aA_{st}b$ , we get the same result.

## Proof (4)

We use *DK*-test to show that  $G$  is deterministic.

- ▶ We need to analyze how  $P$  operates on valid strings by extending its input alphabet and transition function to **process non-terminals** in addition to terminal symbols.
- ▶ Add all symbols  $A_{pq}$  to  $P$ 's input alphabet and extend its  $\delta$  by defining

$$\delta(p, A_{pq}, \varepsilon) = (q, \varepsilon).$$

Set all other transitions involving  $A_{pq}$  to  $\emptyset$ .

- ▶ To preserve  $P$ 's deterministic behavior, if  $P$  reads  $A_{pq}$  from the input then disallow an  $\varepsilon$ -move.

## Proof (5)

Consider the derivation:

$$A_{q_0, q_{\text{accept}}} = v_0 \Rightarrow v_1 \Rightarrow \cdots \Rightarrow v_i \Rightarrow \cdots \Rightarrow v_k = w.$$

### Claim

*If  $P$  reads  $v_i$  containing a non-terminal  $A_{pq}$ , it enters state  $p$  just prior to reading  $A_{pq}$ .*

If  $i = 0$ , then  $v_i = A_{q_0, q_{\text{accept}}}$  and  $P$  starts in  $q_0$ .

Assume the claim is true for some  $i \geq 1$ .

- $v_i = xA_{pq}y$  and  $A_{pq}$  is the non-terminal substituted in the step  $v_i \Rightarrow v_{i+1}$ . By IH,  $P$  enters state  $p$  after reading  $x$ , prior to reading  $A_{pq}$ . By the construction of  $G$ , the substitution rules may of two types:
  1.  $A_{pq} \rightarrow A_{pr}aA_{st}b$  or
  2.  $A_{pp} \rightarrow \varepsilon$ .

## Proof (6)

- ▶ Thus either  $v_{i+1} = xA_{pr}aA_{st}by$  or  $v_{i+1} = xy$ .
- ▶ In the first case, when  $P$  reads  $A_{pr}aA_{st}b$  in  $v_{i+1}$ , we know it starts in state  $p$ , because it has just finished reading  $x$ .

As  $P$  reads  $A_{pr}aA_{st}b$  in  $v_{i+1}$ , it enters the sequence of states  $r, s, t$ , and  $q$ , due to the substitution rule's construction.

- ▶ Therefore, it enters state  $p$  just prior to reading  $A_{pr}$  and it enters state  $s$  just prior to reading  $A_{st}$ .
- ▶ The claim holds on non-terminals in the  $y$  part because, after reading  $b$ ,  $P$  enters state  $q$  and then it reads  $y$  ( $A_{pq} \rightarrow A_{pr}aA_{st}b$ ). On input  $v_i$ , it also enters  $q$  just before reading  $y$ , so the computations agree on the  $y$  parts of  $v_i$  and  $v_{i+1}$ .

Obviously, the computations agree on the  $x$  parts.

- ▶ In the second case, no new non-terminals are introduced, so we only need to observe that the computations agree on the  $x$  and  $y$  parts of  $v_i$  and  $v_{i+1}$ .



# Proof (7)

## Claim

*G passes the DK-test.*

Select one of the accept states of *DK* for *G*. It contains a completed rule *R* which can be

1.  $A_{pq} \rightarrow A_{pr}aA_{st}b.$  or
2.  $A_{pp} \rightarrow .$

We need to show in both cases that the state cannot contain

- a. another complete rule, and
- b. a dotted rule that has a terminal symbol immediately after the dot.

In each case, we start by considering a string *z* on which *DK* goes to the above accept state.

## Case 1a.

Suppose  $R$  is a completed type 1-2 rule:  $A_{pq} \rightarrow A_{pr}aA_{st}b$ . For any rule in this accept state,  $z$  must end with  $A_{pr}aA_{st}b$  because  $DK$  goes to that state on  $z$ . Hence the symbols preceding the dot must be consistent in all such rules.

- ▶ These symbols are  $A_{pr}aA_{st}b$  in  $R$  so any other type 1-2 completed rule must have exactly the same symbols on the right-hand side
- ▶ The non-terminal on the left-hand side must also agree, so the rules must be the same.

## Case 1a. (cont'd)

- ▶ Assume that the accept state contains  $R$  and some type 3 completed  $\varepsilon$ -rule:  $T = A_{ss} \rightarrow ..$
- ▶ From  $R$  we know that  $z$  ends with  $A_{pr}aA_{st}b$ .
- ▶  $P$  pops its stack at the end of  $z$  because a pop occurs at that point in  $R$ , due to  $G$ 's construction.  $\delta(t, b, u) = (q, \epsilon)$
- ▶ According to the way we build  $DK$ , a completed  $\varepsilon$ -rule in a state must derive from a dotted rule that resides in the same state, where the dot isn't at the very beginning and the dot immediately precedes some non-terminals.  $A \rightarrow A_{q_1, q_2}x.A_{ss}y$ .
- ▶ An exception occurs at  $DK$ 's start state, where this dot may occur at the beginning of the rule, but this accept state cannot be the start state because it contains a completed type 1-2 rule.
- ▶ In  $G$ , that means  $T$  derives from a type 1-2 dotted rule where the dot precedes the second non-terminal. From  $G$ 's construction a push occurs just before the dot.  $\delta(q_2, x, \epsilon) = (s, u')$
- ▶ This implies that (**deterministic**) PDA  $P$  does a push move at the very end of  $z$ , contradicting our previous statement.