香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF DATA SCIENCE
數據科學學院

CSC 6011 - Theory of Computation - Lecture 4: Context Free Grammar, Pushdown Automata

Time: Mon & Wed, 4:00 - 5:20 PM

Location: Teaching A, 107

Instructor: Tao LIN (林涛)

TA: Yefan GAO (高叶繁)

Fall 2026

Lecture 4

Last time:

- Finite automata → regular expressions
- Proving languages aren't regular
- Context free grammars

Today:

- Context free grammars (CFGs) – definition
- Context free languages (CFLs)
- Pushdown automata (PDA)
- Converting CFGs to PDAs

Context Free Grammars (CFGs)

G_1

S → 0S1
S → R
R → ε

Shorthand:

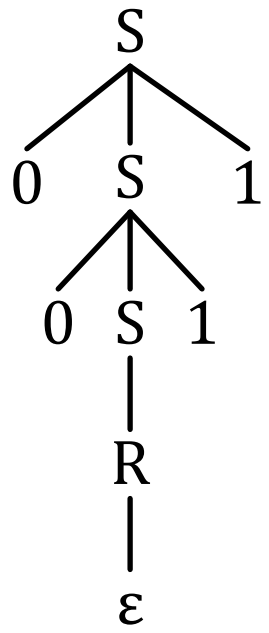
S → 0S1 | R
R → ε

Recall that a CFG has terminals, variables, and rules.

Grammars generate strings

1. Write down start variable
2. Replace any variable by RHS according to a rule
   Repeat until only terminals remain
3. Result is the generated string
4. $L(G)$ is the language of all generated strings
5. We call $L(G)$ a Context Free Language.

Example of G_1 generating a string

Parse tree	Resulting string
	S 0S1 00S11 00R11 $0011 \in L(G_1)$

$$L(G_1) = \{0^k 1^k \mid k \geq 0\}$$

CFG – Formal Definition

Defn: A Context Free Grammar (CFG) G is a 4-tuple (V, Σ, R, S)

V finite set of variables

Σ finite set of terminal symbols

R finite set of rules of the form $V \to (V \cup \Sigma)^*$

S start variable

For $u, v \in (V \cup \Sigma)^*$ write

1) $u \Rightarrow v$ (u "yields" v) if can go from u to v with one substitution step in G

2) $u \overset{*}{\Rightarrow} v$ (u "derives" v) if can go from u to v with some number of substitution steps in G

$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k = v$ is called a derivation of v from u.

If $u = S$ then it is a derivation of v.

$L(G) = \{w | \, w \in \Sigma^* \text{ and } S \overset{*}{\Rightarrow} w\}$

Defn: A is a Context Free Language (CFL) if $A = L(G)$ for some CFG G.

CFG – Formal Definition

Defn: A Context Free Grammar (CFG) G is a 4-tuple (V, Σ, R, S)

V finite set of variables

Σ finite set of terminal symbols

R finite set of rules of the form $V \to (V \cup \Sigma)^*$

S start variable

For $u, v \in (V \cup \Sigma)^*$ write

1) $u \Rightarrow v$ (u "yields" v) if can go from u to v wi

2) $u \overset{*}{\Rightarrow} v$ (u "derives" v) if can go from u to v wi

$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k = v$ is calle

If $u = S$ then it is a derivation of v.

$L(G) = \{w | \, w \in \Sigma^* \text{ and } S \overset{*}{\Rightarrow} w\}$

Defn: A is a Context Free Language (CFL) if $A = L(G)$ for some CFG G.

Mini-Quiz 4.1

Which of these are valid CFGs?

C_1: B → 0B1 | ε
B1 → 1B
0B → 0B

C_2: S → 0S | S1
R → RR

a) C_1 only

b) C_2 only

c) Both C_1 and C_2

d) Neither

CFG – Formal Definition

Defn: A Context Free Grammar (CFG) G is a 4-tuple (V, Σ, R, S)

V finite set of variables

Σ finite set of terminal symbols

R finite set of rules of the form $V \to (V \cup \Sigma)^*$

S start variable

For $u, v \in (V \cup \Sigma)^*$ write

1) $u \Rightarrow v$ (u "yields" v) if can go from u to v wi

2) $u \overset{*}{\Rightarrow} v$ (u "derives" v) if can go from u to v wi

$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k = v$ is calle

If $u = S$ then it is a derivation of v.

$L(G) = \{w | \, w \in \Sigma^* \text{ and } S \overset{*}{\Rightarrow} w\}$

Defn: A is a Context Free Language (CFL) if $A = L(G)$ for some CFG G.

Mini-Quiz 4.1

Which of these are valid CFGs? $L(C_2) = \emptyset$

C_1: B → 0B1 | ε
B1 → 1B
0B → 0B

C_2: S → 0S | S1
R → RR

C_1 is a "Context Sensitive Grammar"

a) C_1 only

b) C_2 only

c) Both C_1 and C_2

d) Neither

CFG – Example

G_2

E → E+T | T

T → T×F | F

F → (E) | a

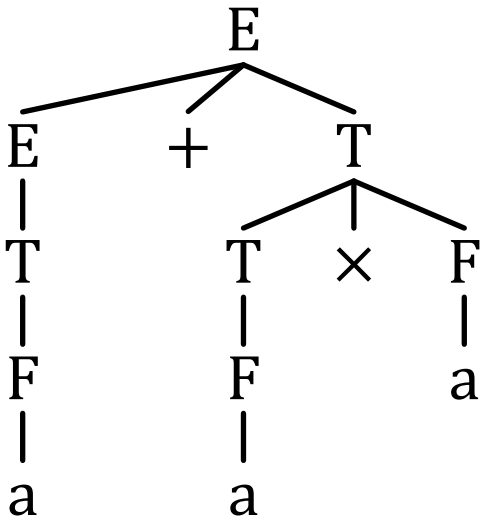
$V =$ {E, T, F}

$\Sigma =$ {+, ×, (,), a}

$R =$ the 6 rules above

$S =$ E

Parse tree



Resulting string

$$E$$

$$E + T$$

$$T + T \times F$$

$$F + F \times a$$

$$a + a \times a \in L(G_2)$$

Generates a+a×a , (a+a)×a, a, a+a+a, etc.

Observe that the parse tree contains additional information, such as the precedence of × over + .

If a string has two different parse trees then it is derived ambiguously and we say that the grammar is ambiguous.

Ambiguity

G_2

E → E+T | T

T → T×F | F

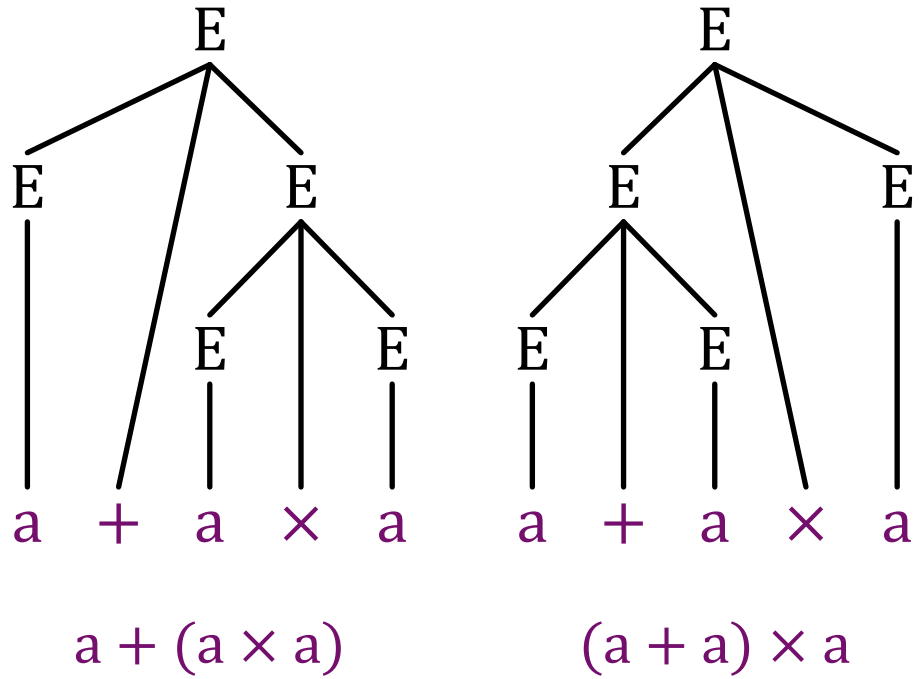
F → (E) | a

G_3

E → E+E | E×E | (E) | a

Both G_2 and G_3 recognize the same language, i.e., $L(G_2) = L(G_3)$.

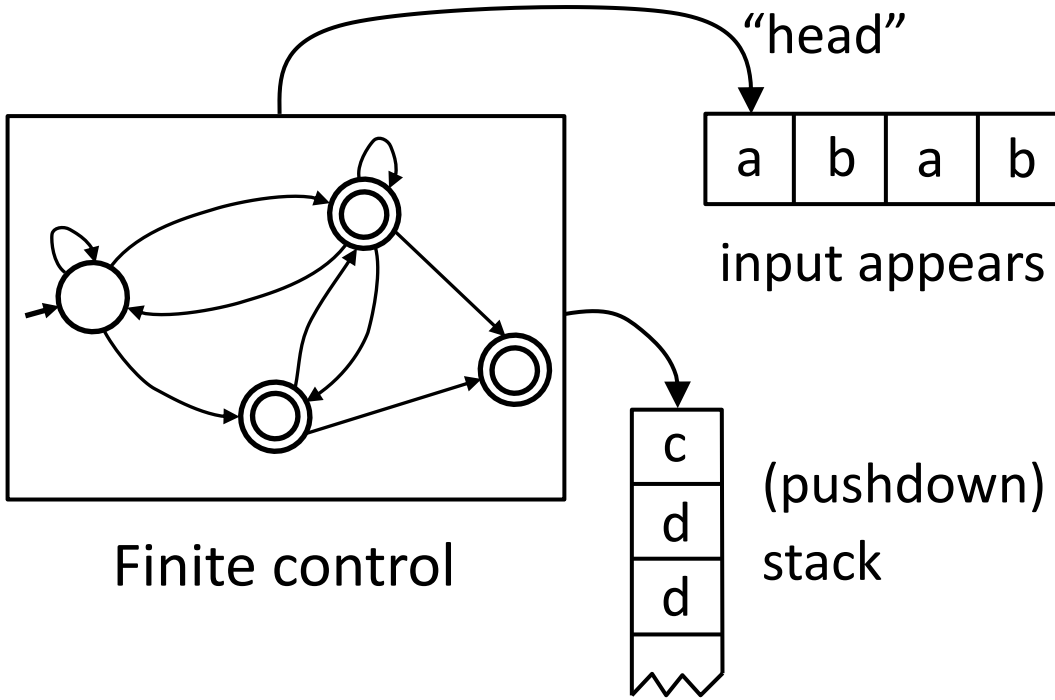
However, G_2 is an unambiguous CFG while G_3 is ambiguous.



a + (a × a)

(a + a) × a

Pushdown Automata (PDA) (Automata for CFL)



Operates like an NFA except can write-add or read-remove symbols from the top of stack.

"push" "pop"

Example: PDA for $D = \{0^k 1^k \mid k \geq 0\}$

1) Read 0s from input, push onto stack until read 1.
2) Read 1s from input, while popping 0s from stack.
3) Enter accept state if stack is empty.

   (note: acceptance only at end of input)

PDA – Formal Definition

Defn: A Pushdown Automaton (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$

Σ input alphabet

Γ stack alphabet

δ: $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$

$\delta(q, \mathrm{a}, \mathrm{c}) = \{(r_1, \mathrm{d}),\ (r_2, \mathrm{e})\}$

Accept if some "thread" is in an accept state at the end of the input string.

Example: PDA for $B = \{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$

Sample input:

0	1	1	1	1	0

1) Read and push input symbols.
   Nondeterministically either repeat or go to (2).
2) Read input symbols and pop stack symbols, compare.
   If ever ≠ then thread rejects.
3) Enter accept state if stack is empty. (Assume that the PDA can check if the stack is empty)

The nondeterministic forks replicate the stack.

This language requires nondeterminism.

Our PDA model is nondeterministic.

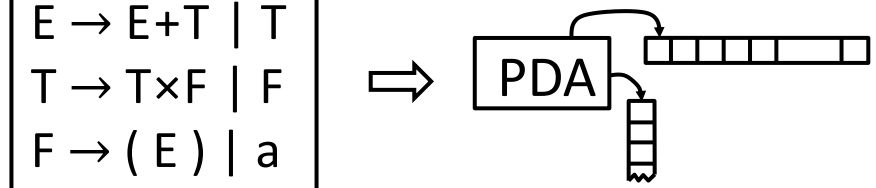
CFGs = PDAs : Converting CFGs to PDAs

Theorem: If A is a CFL, then some PDA recognizes A

Proof: Convert A's CFG to a PDA

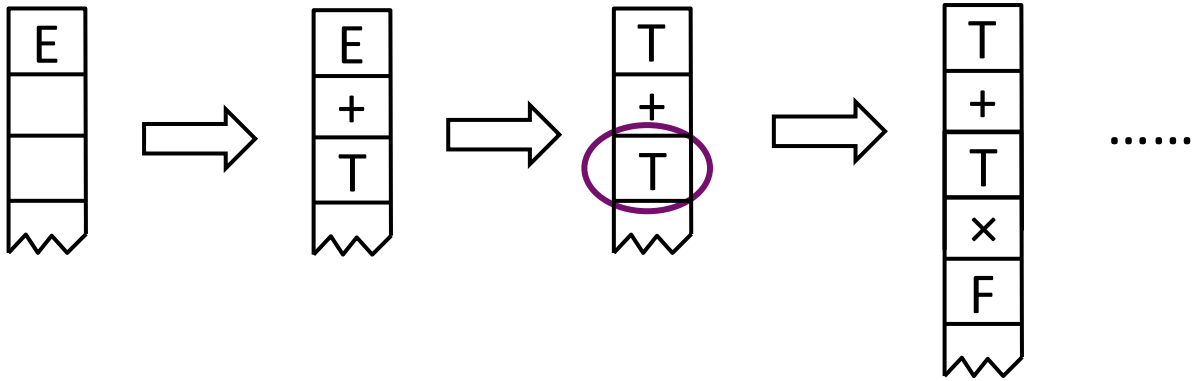
CFG

$E \to E+T \mid T$
$T \to T\times F \mid F$
$F \to (E) \mid a$

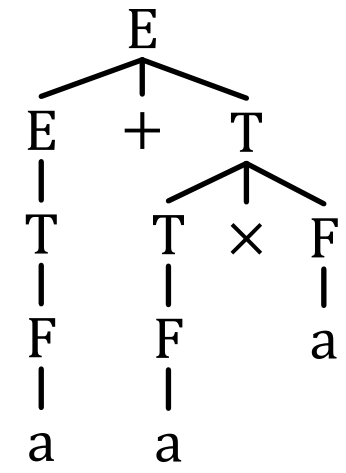


IDEA:
Let PDA begin with starting variable and guess substitutions.
It keeps intermediate generated strings on stack.
When done, compare with input.

Input: a | + | a | × | a



E
E+T
T+T×F
F+F×a
a+a×a



Problem: we cannot access below the top of stack!

Instead, only substitute variables when on the top of stack.

If a terminal is on the top of stack, pop it and compare with input. Reject if ≠.

Converting CFGs to PDAs (continued)

Theorem: If A is a CFL, then some PDA recognizes A

Proof construction: Convert the CFG for A to the following PDA.

1) Push the start symbol on the stack.
2) If the top of stack is

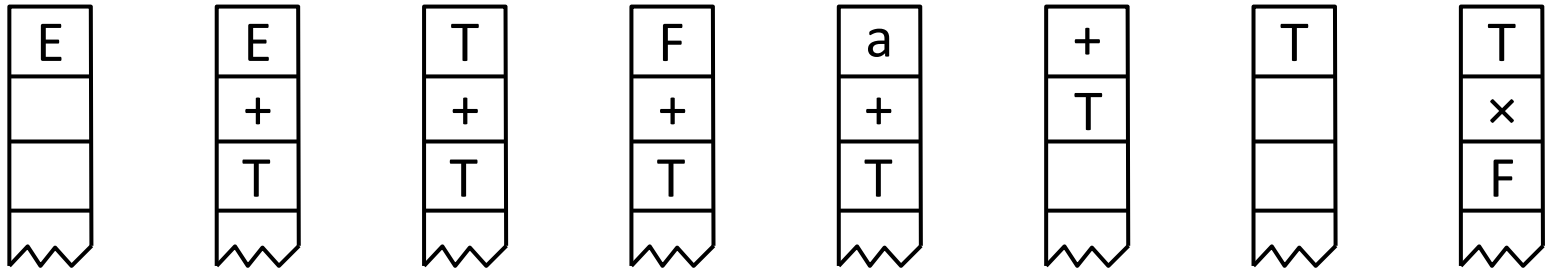
   Variable: replace with right hand side of rule (nondet choice).

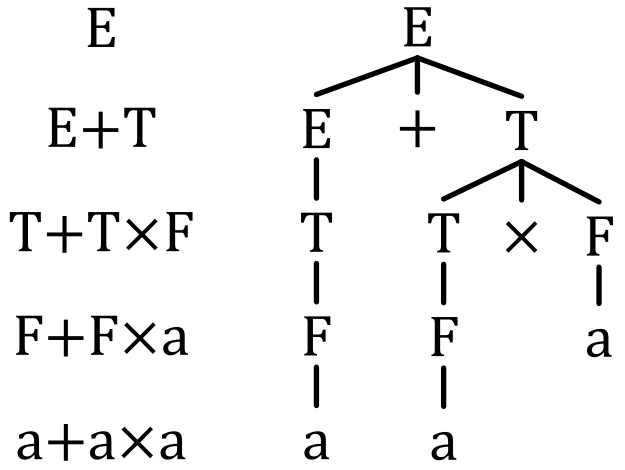
   Terminal: pop it and match with next input symbol.
3) If the stack is empty, *accept.*

$E \rightarrow E+T \mid T$
$T \rightarrow T\times F \mid F$
$F \rightarrow (E) \mid a$

a	+	a	×	a

Example:





Equivalence of CFGs and PDAs

Theorem: A is a CFL iff* some PDA recognizes A

* "iff" = "if and only if" means the implication goes both ways: forward (→) and reverse (←)

CFG → PDA Proved.

PDA → CFG You should know this direction is true. The proof is not required.

Mini-Quiz 4.3

Is every Regular Language also a Context Free Language?

(a) Yes

(b) No

(c) Not sure

Equivalence of CFGs and PDAs

Theorem: A is a CFL iff* some PDA recognizes A

* "iff" = "if and only if" means the implication goes both ways: forward (→) and reverse (←)

CFG → PDA Proved.

PDA → CFG You should know this direction is true. The proof is not required.

Mini-Quiz 4.3

Is every Regular Language also a Context Free Language?

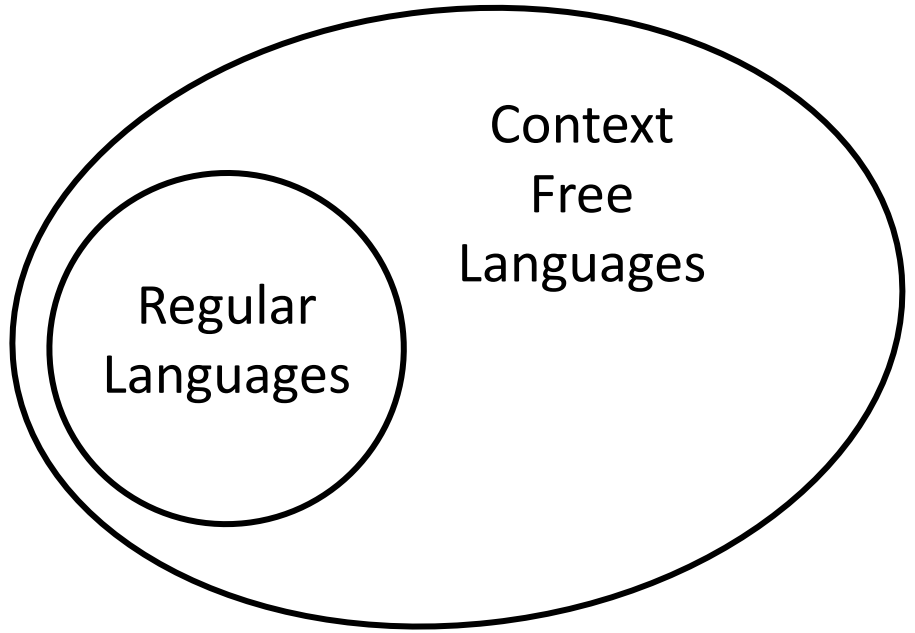
(a) Yes

(b) No

(c) Not sure

Recap

	Recognizer	Generator
Regular Language	DFA or NFA	Regular Expression
Context Free Language	PDA	Context Free Grammar



Quick review of today

1. Defined Context Free Grammars (CFGs) and Context Free Languages (CFLs)
2. Defined Pushdown Automata (PDAs)
3. Gave conversion of CFGs to PDAs.

These slides are adapted from Michael Sipser, *18.404J Theory of Computation*, Fall 2020. Massachusetts Institute of Technology:

- Source: MIT OpenCourseWare https://ocw.mit.edu