18.404/6.840 Lecture 4

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

Today: (Sipser §2.2)
- 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 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

Tree of substitutions "parse tree" | S | S | Resulting string

$\in L(G_1)$

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

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 **(**rule form: $V \rightarrow (V \cup \Sigma)^*$ **)**
- S start variable

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

1) $u \Rightarrow v$ if can go from u to v with one substitution step in
2) $u \overset{*}{\Rightarrow} v$ if can go from u to v with some number of substit

$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k = v$ is called a derivati

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

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

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

Check-in 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

Check-in 4.1

CFG – Example

G_2

E → E+T | T

T → T×F | F

F → (E) | a

Parse tree E E Resulting string

$V =$ {E, T, F}
$\Sigma =$ {+, ×, (,), a}
$R =$ the 6 rules above
$S =$ E

Generates a+a×a

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

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

Check-in 4.2

How many reasonable distinct meanings does the following English sentence have?

The boy saw the girl with the mirror.

(a) 1
(b) 2
(c) 3 or more

Check-in 4.2

Ambiguity

G_2

E → E+T | T

T → T×F | F

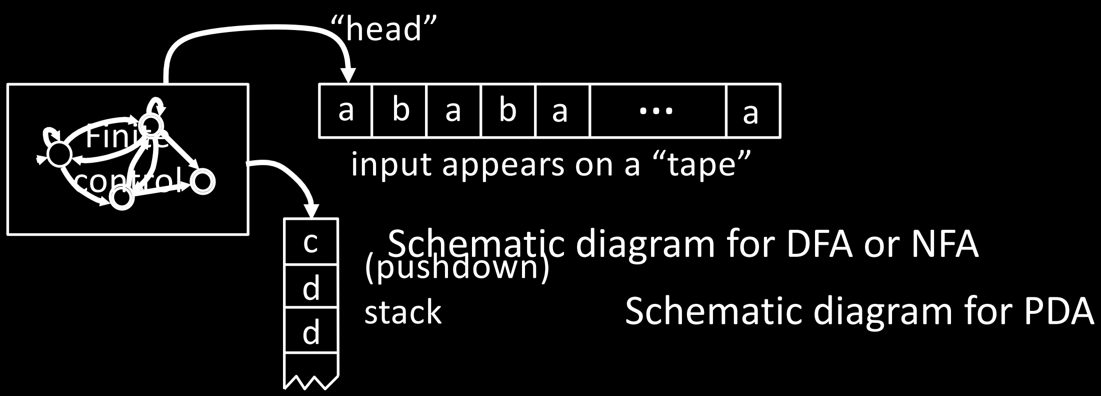
F → (E) | a

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 and G_3 is ambiguous.

Pushdown Automata (PDA)



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

push

pop

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

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 <u>Pushdown Automaton</u> (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$

Σ input alphabet

Γ stack alphabet

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

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

Accept if some thread is in the 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 $\neq$ then thread rejects.
3) Enter accept state if stack is empty. (do in "software")

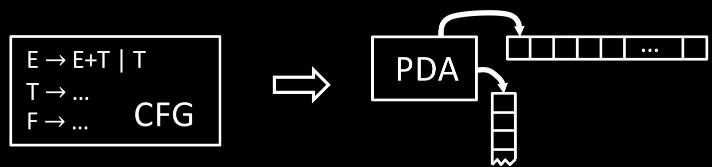
The nondeterministic forks replicate the stack.

This language requires nondeterminism.
Our PDA model is nondeterministic.

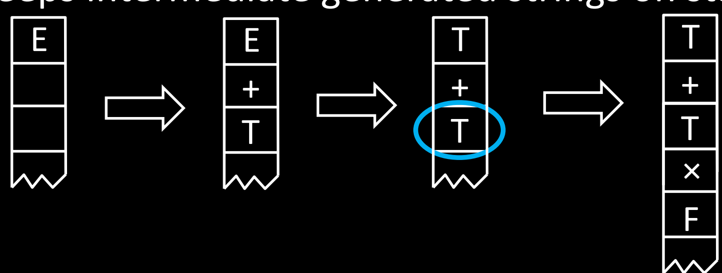
Converting CFGs to PDAs

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

Proof: Convert A's CFG to a PDA



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



Input: | a | + | a | × | a |

G_2

E → E+T | T

T → T×F | F

F → (E) | a

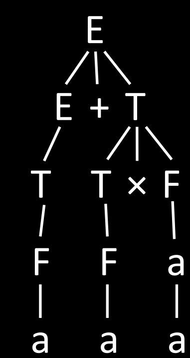
E

E+T

T+T×F

F+F×a

a+a×a



Problem! Access below the top of stack is cheating!

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 (contd)

G_2
E → E+T | T
T → T×F | F
F → (E) | a

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

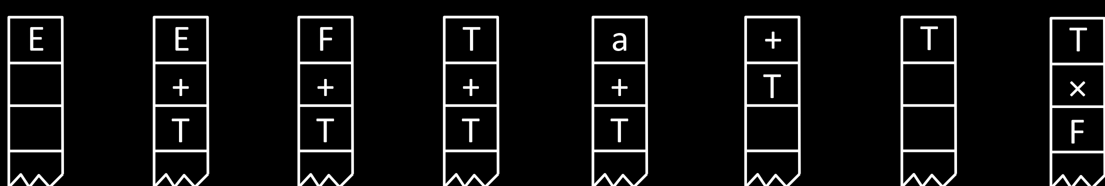
Proof construction: Convert the CFG for A to the following PDA.

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

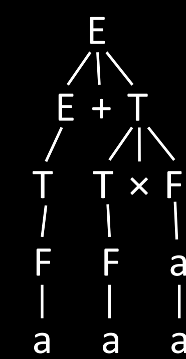
   Variable: replace with right hand side of rule (nondet choice).

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

Example: a + a × a



E
E+T
T+T×F
F+F×a
a+a×a



Equivalence of CFGs and PDAs

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

⟷ Done.

In book. You are responsible for knowing it is true, but not for knowing the proof.

* "iff" = "if an only if" means the implication goes both ways.
So we need to prove both directions: forward (→) and reverse (←).

Check-in 4.3

Is every Regular Language also a Context Free Language?

(a) Yes

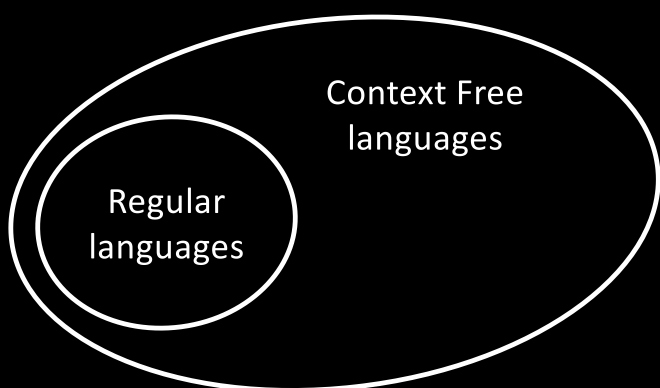
(b) No

(c) Not sure

Check-in 4.3

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.