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"

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: $V \to (V \cup \Sigma)^*$ **)**
- 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 G
2) $u \overset{*}{\Rightarrow} v$ 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.

Check-in 4.1

CFG – Example

G_2

E → E+T | T

T → T×F | F

F → (E) | a

Parse tree

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 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.

Check-in 4.2

Ambiguity

G_2

E → E+T | T

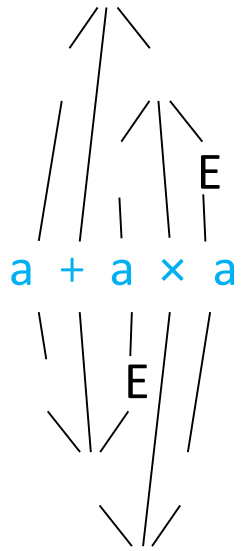
T → T×F | F

F → (E) | a

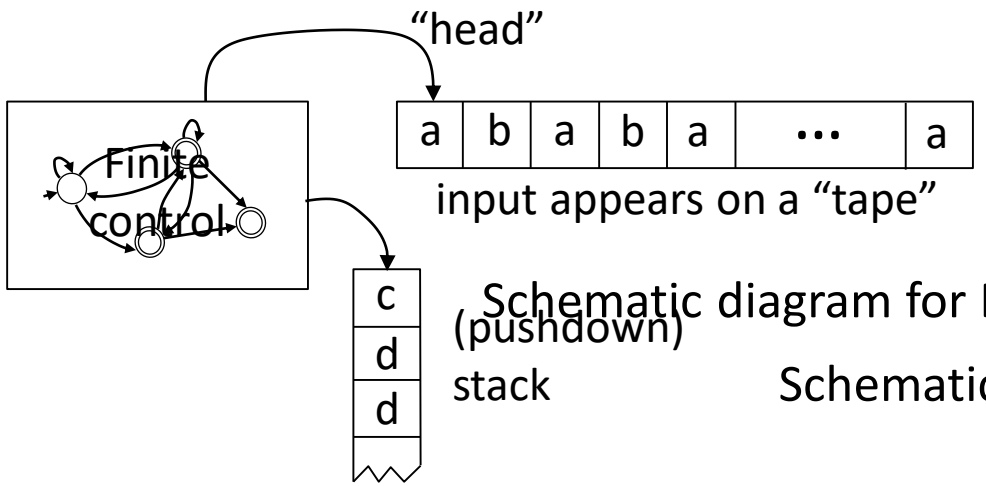
G_3

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)



Schematic diagram for DFA or NFA

Schematic diagram for PDA

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

push pop

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

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

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

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

Example: PDA for $B = \{ww^{\mathcal{R}} | \ w \in \{0,1\}^* \}$ 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. (do in "software")

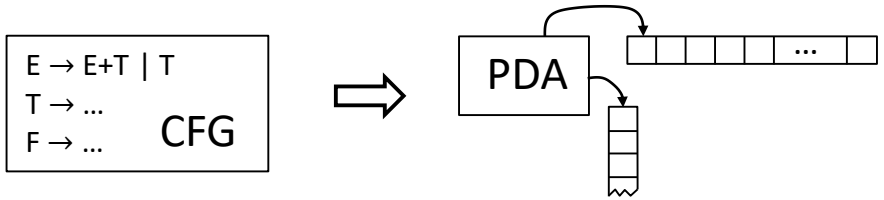
The nondeterministic forks replicate the stack.

This language requires nondeterminism.
Our PDA model is nondeterministic.

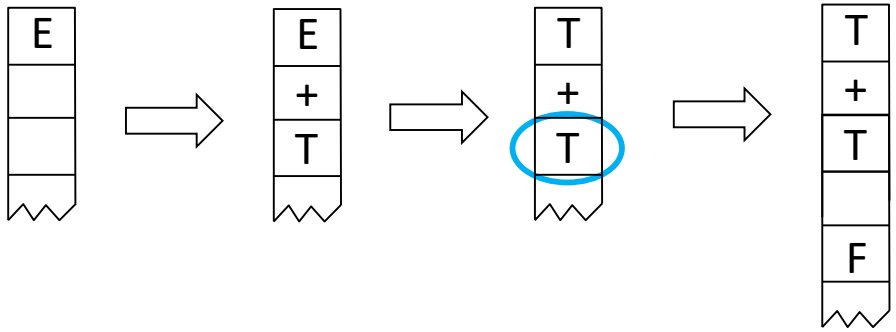
Converting CFGs to PDAs

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.

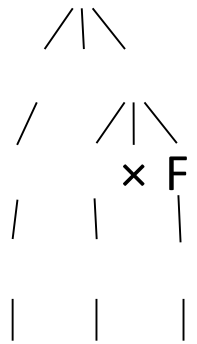


Input:

a	+	a	×	a

G_2

E → E+T | T
T → T×F | F
F → (E) | a



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

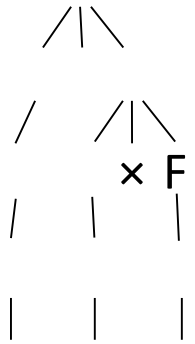
Theorem: If A is a CFL then some PDA recognizes A

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

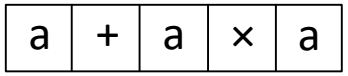
1) Push the start symbol on the stack.
2) If the top of stack is

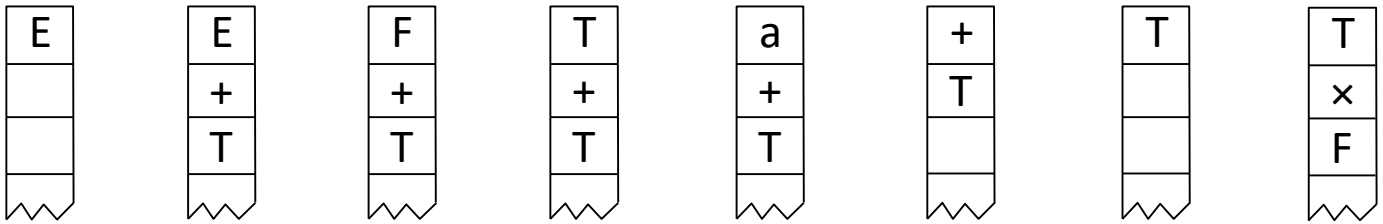
   Variable: replace with right hand side of rule (nondet choice).

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



Example:





Equivalence of CFGs and PDAs

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

$\longleftrightarrow$ 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 ($\rightarrow$) and reverse ($\leftarrow$).

Check-in 4.3

Is every Regular Language also a Context Free Language?

(a) Yes

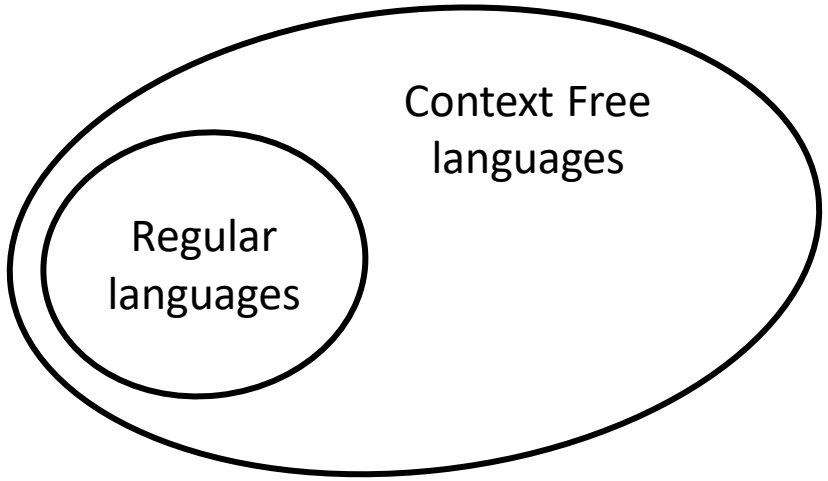
(b) No

(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

3. Gave conversion of CFGs to PDAs.

18.404J Theory of Computation

Fall 2020

18.404/6.840 Lecture 5

Last time:

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

Today: (Sipser §2.3, §3.1)

- Proving languages not Context Free
- Turing machines
- T-recognizable and T-decidable languages

Equivalence of CFGs and PDAs

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

→ Done.

← Need to know the fact, not the proof

Corollaries:

1) Every regular language is a CFL.
2) If A is a CFL and B is regular then $A \cap B$ is a CFL.

Proof sketch of (2):

While reading the input, the finite control of the PDA for A simulates the DFA for B.

Note 1: If A and B are CFLs then $A \cap B$ may not be a CFL (will show today).
Therefore the class of CFLs is not closed under $\cap$.

Note 2: The class of CFLs is closed under $\cup, \circ, *$ (see Pset 2).

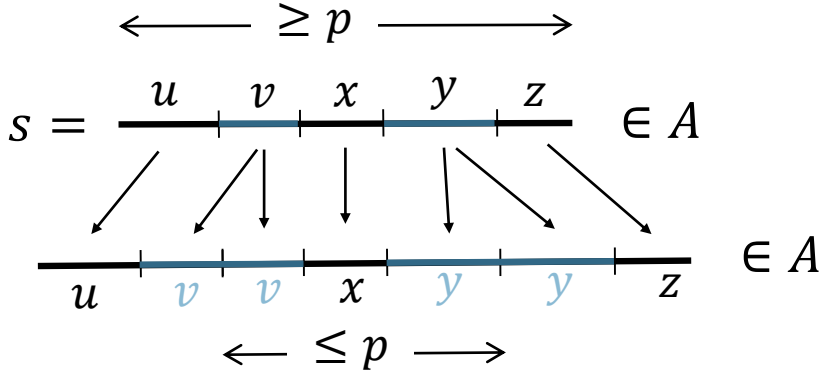
Proving languages not Context Free

Let $B = \{0^k 1^k 2^k \mid k \geq 0\}$. We will show that B isn't a CFL.

Pumping Lemma for CFLs: For every CFL A, there is a p such that if $s \in A$ and $|s| \geq p$ then $s = uvxyz$ where

1) $uv^i xy^i z \in A$ for all $i \geq 0$
2) $vy \neq \varepsilon$
3) $|vxy| \leq p$

Informally: All long strings in A are pumpable and stay in A.

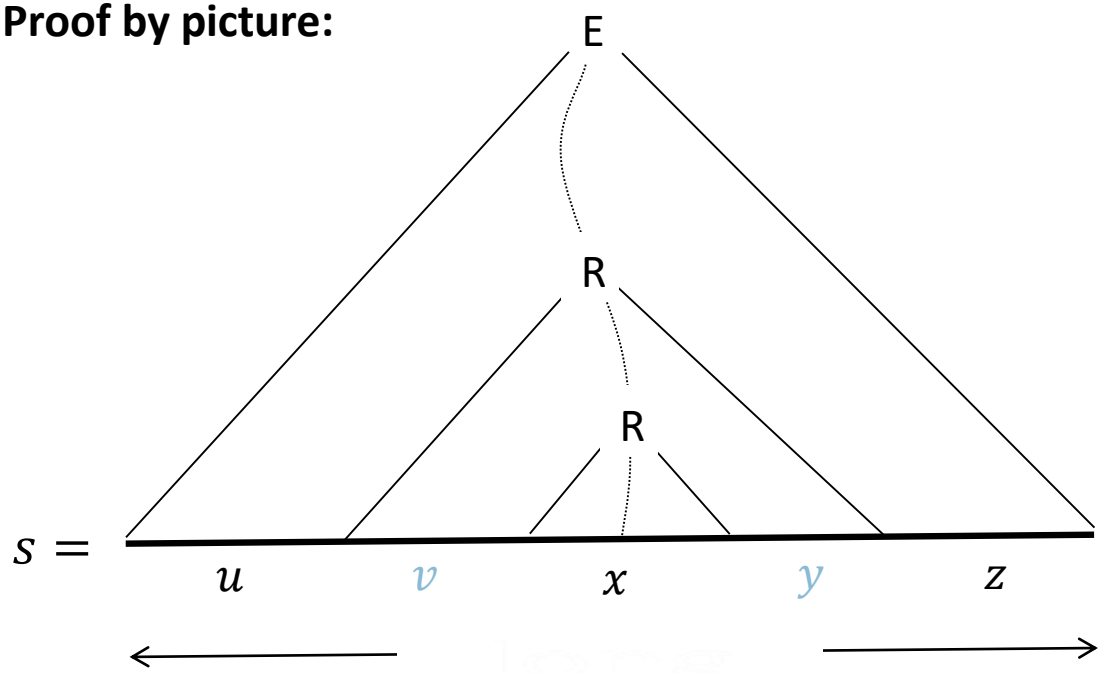


Pumping Lemma – Proof

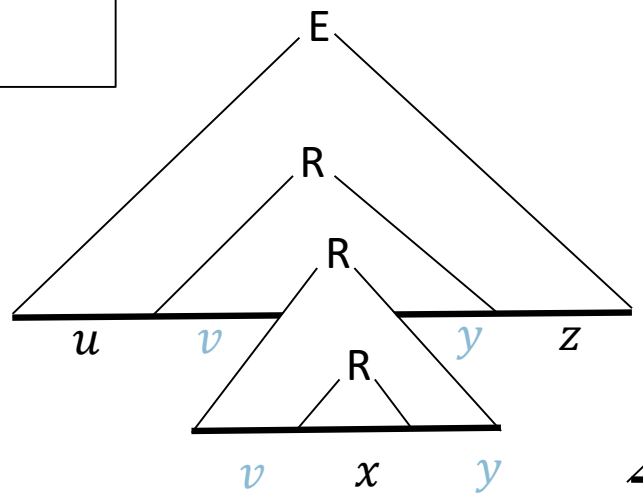
Pumping Lemma for CFLs: For every CFL A, there is a p such that if $s \in A$ and $|s| \geq p$ then $s = uvxyz$ where

1) $uv^ixy^iz \in A$ for all $i \geq 0$
2) $vy \neq \varepsilon$
3) $|vxy| \leq p$

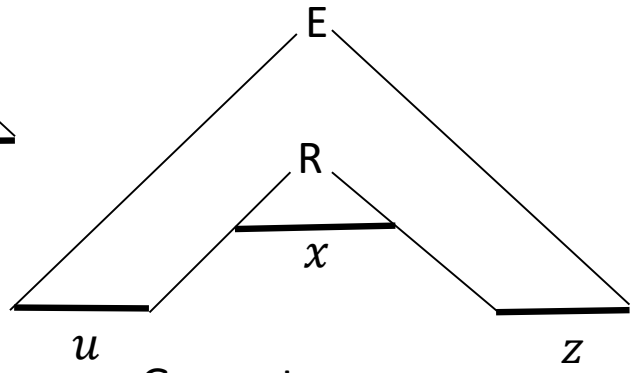
Proof by picture:



Long $s \rightarrow$ tall parse tree



Generates $uvvxyyz = uv^2xy^2z$



Generates $uxz = uv^0xy^0z$

"cutting and pasting" argument

Pumping Lemma – Proof details

For $s \in A$ where $|s| \geq p$, we have $s = uvxyz$ where:
1) $uv^i xy^i z \in A$ for all $i \geq 0$
2) $vy \neq \varepsilon$
3) $|vxy| \leq p$

Let b = the length of the longest right hand side of a rule (E → E+T)
= the max branching of the parse tree

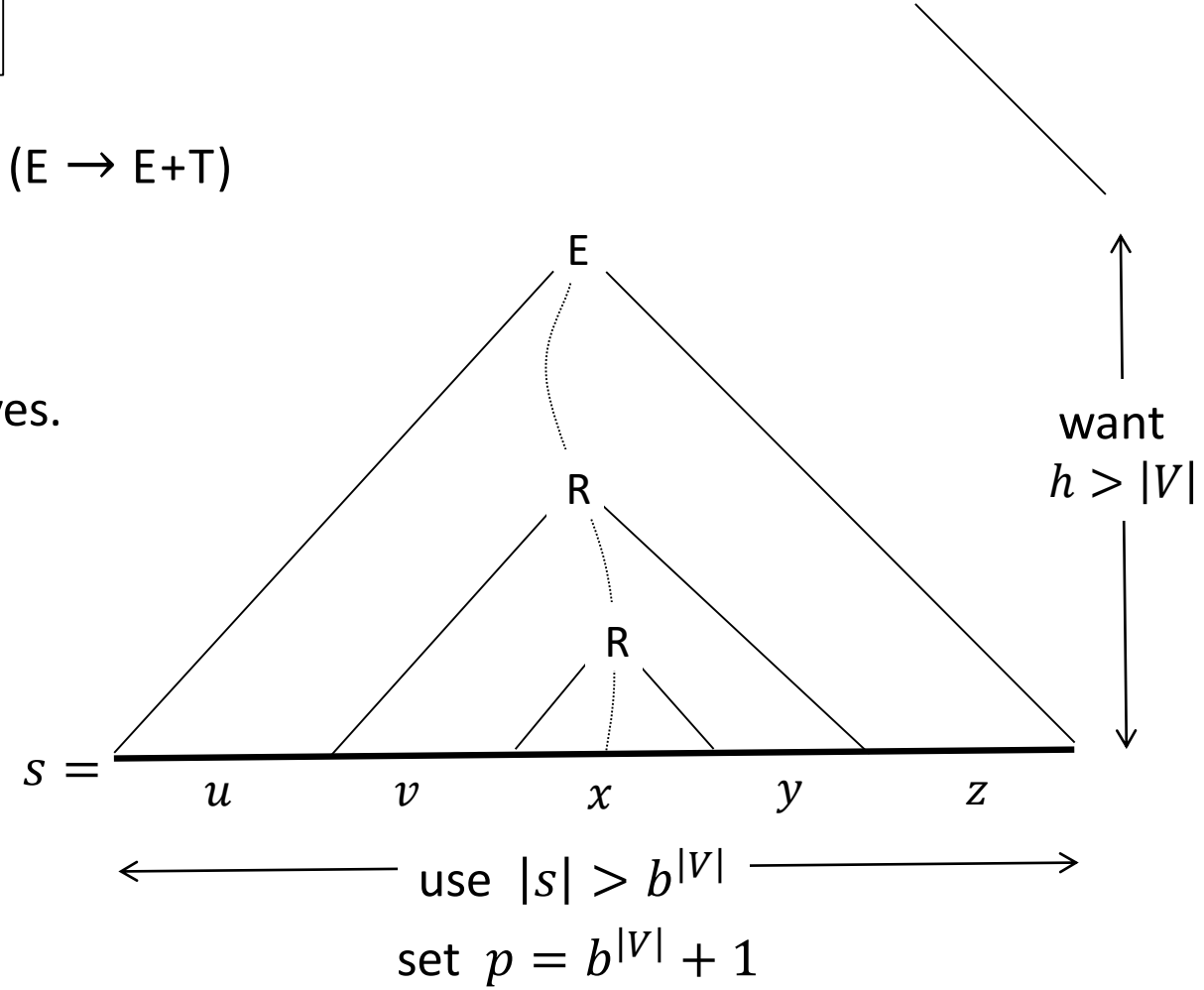
Let h = the height of the parse tree for s.

A tree of height h and max branching b has at most b^h leaves.
So $|s| \leq b^h$.

Let $p = b^{|V|} + 1$ where $|V|$ = # variables in the grammar.

So if $|s| \geq p > b^{|V|}$ then $|s| > b^{|V|}$ and so $h > |V|$.

Thus at least $|V| + 1$ variables occur in the longest path.
So some variable R must repeat on a path.



Example 1 of Proving Non-CF

Pumping Lemma for CFLs: For every CFL A, there is a p such that if $s \in A$ and $|s| \geq p$ then $s = uvxyz$ where

1) $uv^i xy^i z \in A$ for all $i \geq 0$
2) $vy \neq \varepsilon$
3) $|vxy| \leq p$

Let $B = \{0^k 1^k 2^k \mid k \geq 0\}$

Show: B is not a CFL

Proof by Contradiction:

Assume (to get a contradiction) that B is a CFL .

The CFL pumping lemma gives p as above. Let $s = 0^p 1^p 2^p \in B$.

Pumping lemma says that can divide $s = uvxyz$ satisfying the 3 conditions.

Condition 3 ($|vxy| \leq p$) implies that vxy cannot contain both 0s and 2s.

So uv^2xy^2z has unequal numbers of 0s, 1s, and 2s.

Thus $uv^2xy^2z \notin B$, violating Condition 1. Contradiction!

Therefore our assumption (B is a CFL) is false. We conclude that B is not a CFL .

$$s = \underbrace{00\cdots0011\cdots1122\cdots22}_{\begin{array}{ccccc} u & v & x & y & z \\ \leftarrow & & \leq p & & \rightarrow \end{array}}$$

Check-in 5.1

Example 2 of Proving Non-CF

Pumping Lemma for CFLs: For every CFL A, there is a p such that if $s \in A$ and $|s| \geq p$ then $s = uvxyz$ where

1) $uv^i xy^i z \in A$ for all $i \geq 0$
2) $vy \neq \varepsilon$
3) $|vxy| \leq p$

Let $F = \{ww | w \in \Sigma^*\}$. $\Sigma = \{0,1\}$.

Show: F is not a CFL.

Assume (for contradiction) that F is a CFL.
The CFL pumping lemma gives p as above. Need to choose $s \in F$. Which s?

Try $s_1 = 0^p 1 0^p 1 \in F$.

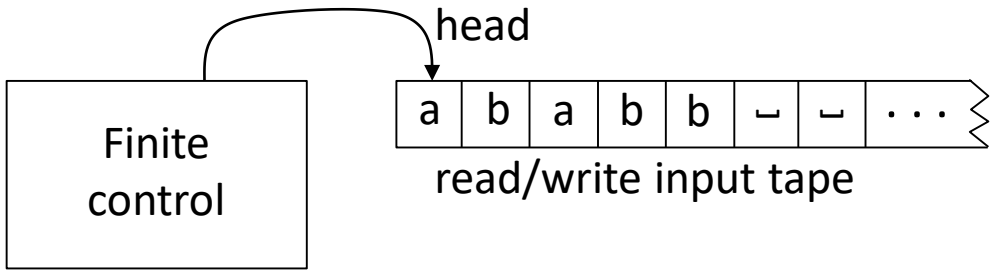
$s_1 = \underbrace{000\cdots001}_{u}\underbrace{000\cdots}_{v}\underbrace{0}_{x}\underbrace{0}_{y}\underbrace{\cdots001}_{z}$, with vxy marked $\leq p$

Try $s_2 = 0^p 1^p 0^p 1^p \in F$.

$s_2 = \underbrace{0\cdots01}_{u}\underbrace{\cdots10}_{v}\underbrace{\cdots}_{x}\underbrace{0}_{y}\underbrace{\cdots01\cdots1}_{z}$, with vxy marked $\leq p$

Show s_2 cannot be pumped $s_2 = uvxyz$ satisfying the 3 conditions.
Condition 3 implies that vxy does not overlap two runs of 0s or two runs of 1s.
Therefore, in uv^2xy^2z, two runs of 0s or two runs of 1s have unequal length.
So $uv^2xy^2z \notin F$ violating Condition 1. Contradiction! Thus F is not a CFL.

Turing Machines (TMs)

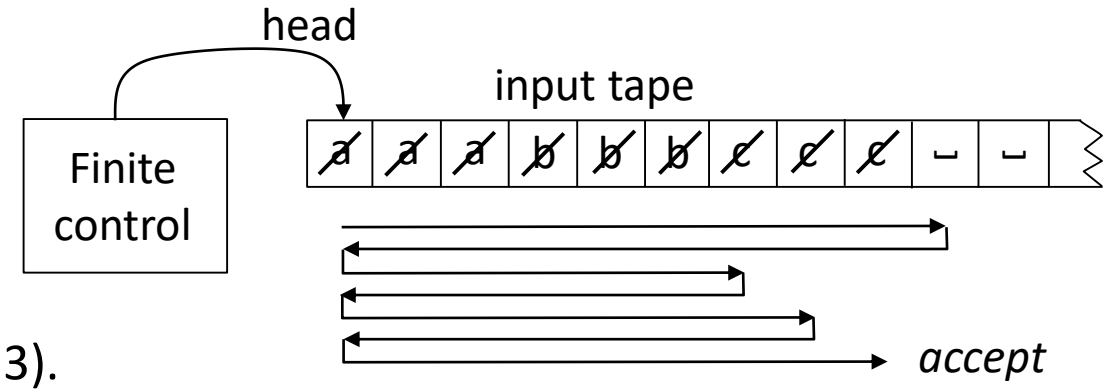


1) Head can read and write
2) Head is two way (can move left or right)
3) Tape is infinite (to the right)
4) Infinitely many blanks "␣" follow input
5) Can accept or reject any time (not only at end of input)

TM – example

TM recognizing $B = \{a^k b^k c^k \mid k \geq 0\}$

1) Scan right until ___ while checking if input is in $a^*b^*c^*$, *reject* if not.
2) Return head to left end.
3) Scan right, crossing off single a, b, and c.
4) If the last one of each symbol, *accept*.
5) If the last one of some symbol but not others, *reject*.
6) If all symbols remain, return to left end and repeat from (3).



Check-in 5.2

How do we get the effect of "crossing off" with a Turing machine?

a) We add that feature to the model.

b) We use a tape alphabet $\Gamma = \{a, b, c, \not{a}, \not{b}, \not{c}, \ \}$.

c) All Turing machines come with an eraser.

Check-in 5.2

TM – Formal Definition

Defn: A Turing Machine (TM) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$

Σ input alphabet

Γ tape alphabet $(\Sigma \subseteq \Gamma)$

δ: $\text{Q} \times \Gamma \to Q \times \Gamma \times \{\text{L, R}\}$ (L = Left, R = Right)

$\delta(q, \text{a}) = (r, \text{b, R})$

On input w a TM M may halt (enter q_{acc} or q_{rej}) or M may run forever ("loop").

So M has 3 possible outcomes for each input w:

1. *Accept* w (enter q_{acc})
2. *Reject* w by halting (enter q_{rej})
3. *Reject* w by looping (running forever)

Check-in 5.3

This Turing machine model is deterministic.
How would we change it to be nondeterministic?

a) Add a second transition function.

b) Change δ to be δ: $\text{Q} \times \Gamma \to \mathcal{P}(\, Q \times \Gamma \times \{\text{L, R}\}\,)$

c) Change the tape alphabet Γ to be infinite.

TM Recognizers and Deciders

Let M be a TM. Then $L(M) = \{w | M \text{ accepts } w\}$.

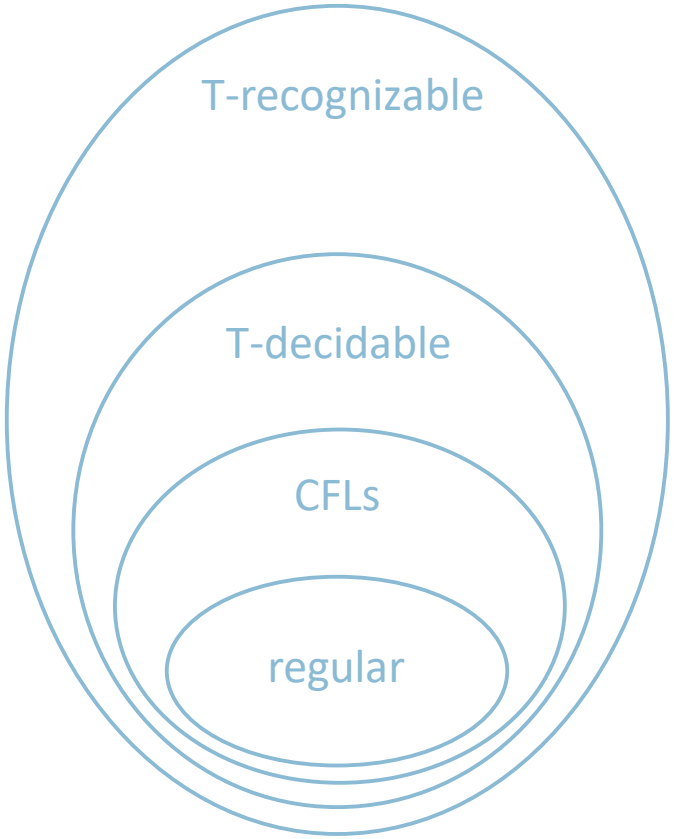
Say that M recognizes A if $A = L(M)$.

Defn: A is Turing-recognizable if $A = L(M)$ for some TM M.

Defn: TM M is a decider if M halts on all inputs.

Say that M decides A if $A = L(M)$ and M is a decider.

Defn: A is Turing-decidable if $A = L(M)$ for some TM decider M.



Quick review of today

4. T-recognizable and T-decidable languages.

18.404J Theory of Computation

Fall 2020