Lecture 13

CSE 431
Intro to Theory of Computation

After Turing → Restricted computing models

McCulloch & Pitts (1943)
Neural Nets (precursor to deep nets)
as models of brain

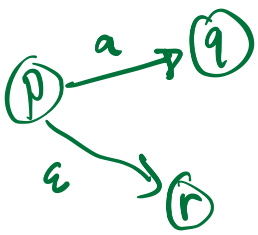
Kleene (1951)
Neural Nets
complicated
≡ Finite State Machines (DFA)
≡ Regular Expressions
defined these and the terms

Chomsky (1956)
Backus-Naur
Chomsky Context-Free languages — as model of human languages
Backus-Naur (aka Backus-Naur Form Grammars)
BNF — for computer languages
also did context-sensitive languages

Rabin & Scott (1959)
Turing Award winning work
Nondeterministic Finite Automata (NFA)
introduced nondeterministic machines
Idea really simplified Kleene's Thm

Following this Claim: CFGs ≡ Pushdown Automata (PDA)
Nondeterministic automata with a stack

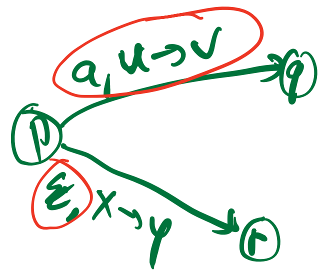
NFA



PDA

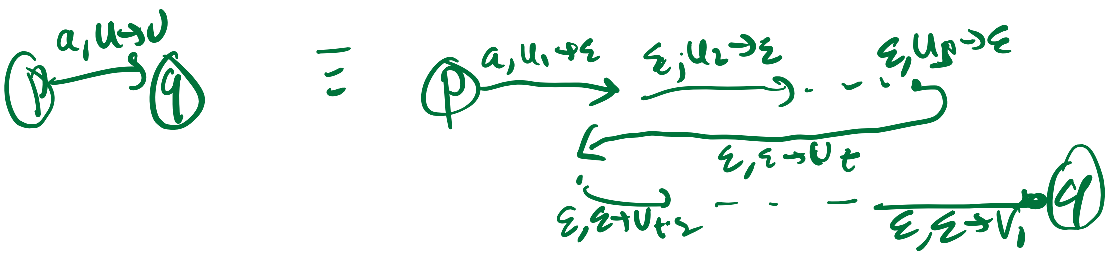
$a \in \Sigma,\ u, v \in \Gamma^*$

Meaning if next input char is an a and top of stack is u can replace u by v and move past a.



Note: Normally we think of only being able to see top symbol on the stack.
but we can convert such a machine into one with that restriction:

Suppose $u = u_1 \cdots u_s \quad v = v_1 \cdots v_t$



Thm For every L, there is a CFG G s.t. $L = L(G)$
$\Leftrightarrow$ there is a PDA M s.t. $L = L(M)$

We will only prove one direction
(The other direction is much less practically important, and is tricky. Sipser's text has the best exposition of it I have seen).

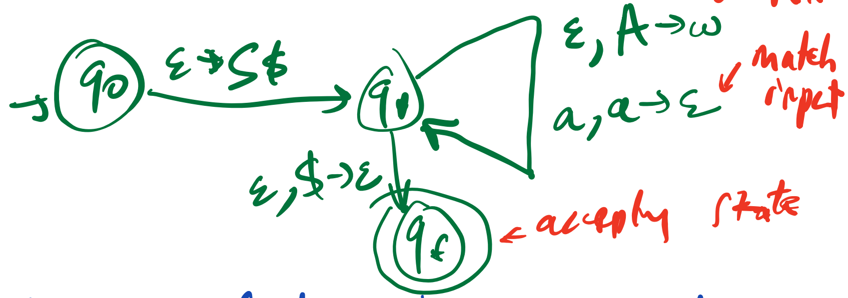
Thm For every CFG G there is a PDA M s.t. $L(G) = L(M)$

Proof: We give two proofs

Top-down parser

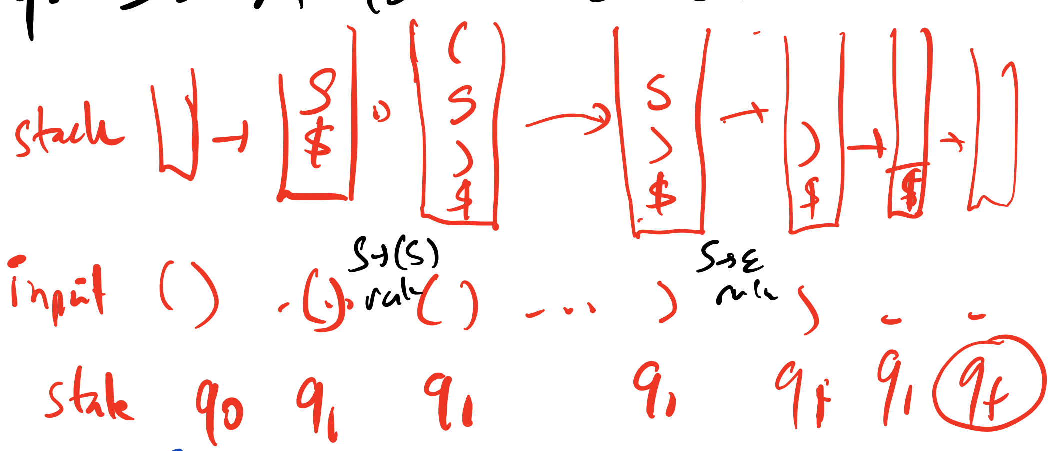
Given CFG G with rules $A \to \omega$ and start symbol S

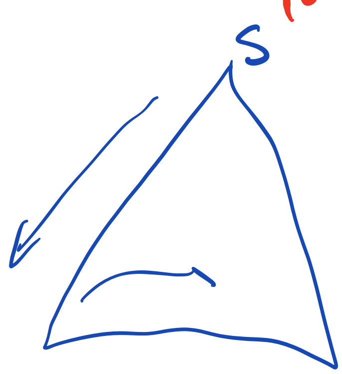
Add new "bottom of stack" symbol $\$$



Idea: M applies rules of derivation to variable at top of the stack, matching up terminals at top of stack to the input

e.g. $S \to (S) \mid SS \mid \varepsilon$ $\qquad S \Rightarrow (S) \Rightarrow ()$





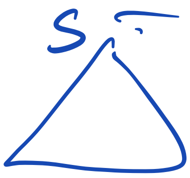
corresponds to top-down DFS of parse tree

∎

Note: This is nondeterministic since a single A may have many rules.

Hard to know which rule to apply since the PDA hasn't even read any of the input symbols that the rule is supposed to generate!

Bottom-up Parser.

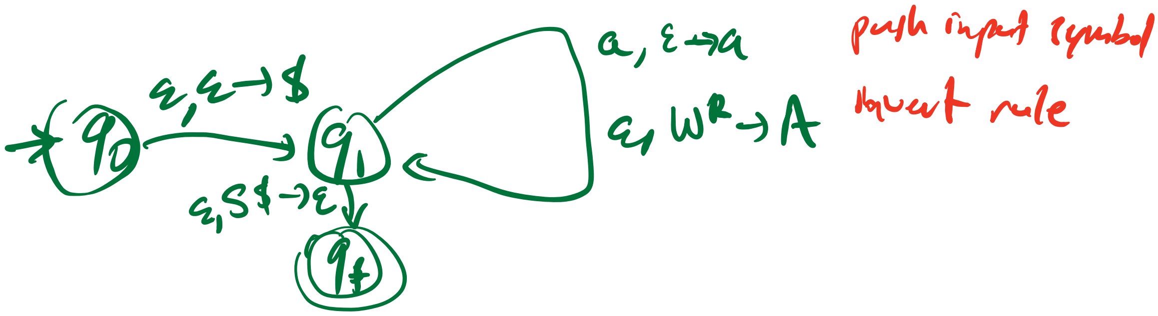


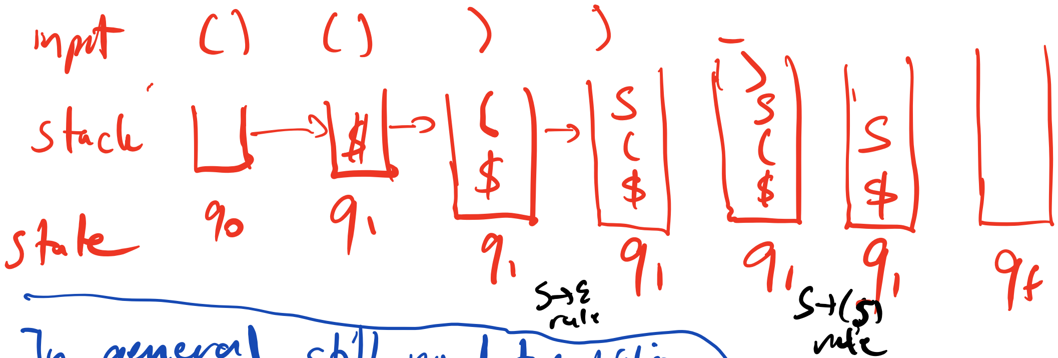
Idea: Push symbols onto the stack and try to invert the CFG derivation to produce S alone on the stack (other than \$.)

Note: If input has symbols $\underline{a_1 a_2 a_3 \cdots a_t} \cdots$ and we start pushing the 1st t symbols one-by-one we get $a_t a_{t-1} \cdots a_1$ on the stack (the last one is on top)

This reverses the string

Bottom-Up parser





In general still nondeterministic and need look-ahead to know which rule to invert.

$)S(\to S$

Programming languages are designed so that one doesn't need to look ahead to know what rule to invert.

Chomsky Normal Form Conversion

rules of form

$A \to BC \qquad B, C \in V,\ B, C \neq S$

$A \to a$

$S \to \varepsilon$

Problems for general rules

4. • $A \to \varepsilon$ for $A \neq S$
2. • Right-hand side of length >1 containing $a \in \Sigma$
3. • Right-hand sides of length >2
   • Rules of form $A \to B$ (unit rules)
1. • S on RHS of a rule

We get rid of these step by step:

$S \to (S) \mid SS \mid \varepsilon$

① Add new S_0 and rule $S_0 \to S$

new start symbol

$$S_0 \to S \quad , \quad S \to (S) \mid SS \mid \varepsilon$$

② Add new var U for each $a \in \Sigma$

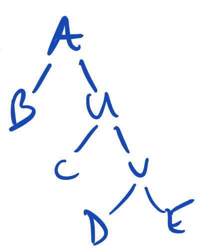
replace a in long right hand sides with U and add rule $U \to a$

$$S_0 \to S \quad , S \to USTISS \mid \varepsilon, U \to (, T \to)$$

③ Rules of length > 2

Add a chain of new intermediate variables to break up into size 2

$\Rightarrow$



$$S_0 \to S, S \to UV \mid SS \mid \varepsilon, U \to (, T \to)$$
$$V \to ST$$

④ Compute $\mathcal{E}$ the set of variables that can produce ε:

If $A \to \varepsilon$ is a rule add A to $\mathcal{E}$

$\mathcal{E} = \{S_0, S\}$

Repeat: if $A \to BC$ is a rule with $B, C \in \mathcal{E}$ add A to $\mathcal{E}$

Add

$S \to S$ → not needed

$V \to T$

$S_0 \to \varepsilon$

(a) if $A \to BC$ where $B \in \mathcal{E}$ add rule $A \to C$

(b) if $A \to BC$ where $C \in \mathcal{E}$ add rule $A \to B$

(c) If $S_0 \in \mathcal{E}$ add rule $S_0 \to \varepsilon$

Any time such a rule is used it can be replaced by (a),(b),(c) rules above

(d) remove all rules $A \to \varepsilon$ for $A \neq S_0$

$$S_0 \to S \mid \varepsilon,\ S \to UV \mid SS \mid S,\ U \to (,\ T \to),\ V \to ST \mid T$$

Note: we added a number of unit rules that might not have been there before

5 Get rid of unit rules:

Create a directed graph on variables where there is an edge

$$A \to B \quad \text{iff} \quad \underset{\text{unit rule}}{A \to B}$$

Note: can do rules that walk around this graph doing replacements but eventually need to do a non-unit rule at one of the vars reachable

$$S_0 \to S \mid \underline{\varepsilon},\ S \to \underline{UV} \mid \underline{SS} \mid S,\ U \to \underline{(},\ T \to \underline{)},\ V \to \underline{ST} \mid T$$

non-unit rules marked.

- Add all non-unit RHS to any var that can reach them
- Remove unit rules

Graph

Final Grammar in Chomsky Normal Form

$$S_0 \to UV \mid SS \mid \varepsilon$$
$$S \to UV \mid SS$$
$$U \to ($$
$$T \to)$$
$$V \to ST \mid)$$