

## Week4 wednesday

Regular sets are not the end of the story

- Many nice / simple / important sets are not regular
- Limitation of the finite-state automaton model: Can't "count", Can only remember finitely far into the past, Can't backtrack, Must make decisions in "real-time"
- We know actual computers are more powerful than this model...

The **next** model of computation. Idea: allow some memory of unbounded size. How?

- To generalize regular expressions: **context-free grammars**
- To generalize NFA: **Pushdown automata**, which is like an NFA with access to a stack: Number of states is fixed, number of entries in stack is unbounded. At each step (1) Transition to new state based on current state, letter read, and top letter of stack, then (2) (Possibly) push or pop a letter to (or from) top of stack. Accept a string iff there is some sequence of states and some sequence of stack contents which helps the PDA process the entire input string and ends in an accepting state.

Is there a PDA that recognizes the nonregular language  $\{0^n 1^n \mid n \geq 0\}$ ?



The PDA with state diagram above can be informally described as:

Read symbols from the input. As each 0 is read, push it onto the stack. As soon as 1s are seen, pop a 0 off the stack for each 1 read. If the stack becomes empty and we are at the end of the input string, accept the input. If the stack becomes empty and there are 1s left to read, or if 1s are finished while the stack still contains 0s, or if any 0s appear in the string following 1s, reject the input.

Trace the computation of this PDA on the input string 01.

Trace the computation of this PDA on the input string 011.

A PDA recognizing the set {  $0^n 1^n$  } can be informally described as:

Read symbols from the input. As each 0 is read, push it onto the stack. As soon as 1s are seen, pop a 0 off the stack for each 1 read. If the stack becomes empty and there is exactly one 1 left to read, read that 1 and accept the input. If the stack becomes empty and there are either zero or more than one 1s left to read, or if the 1s are finished while the stack still contains 0s, or if any 0s appear in the input following 1s, reject the input.

Modify the state diagram below to get a PDA that implements this description:



**Definition** A **pushdown automaton** (PDA) is specified by a 6-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, F)$  where  $Q$  is the finite set of states,  $\Sigma$  is the input alphabet,  $\Gamma$  is the stack alphabet,

$$\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$$

is the transition function,  $q_0 \in Q$  is the start state,  $F \subseteq Q$  is the set of accept states.

Draw the state diagram and give the formal definition of a PDA with  $\Sigma = \Gamma$ .

Draw the state diagram and give the formal definition of a PDA with  $\Sigma \cap \Gamma = \emptyset$ .

*Extra practice:* Consider the state diagram of a PDA with input alphabet  $\Sigma$  and stack alphabet  $\Gamma$ .

Label	means
$a, b; c$ when $a \in \Sigma, b \in \Gamma, c \in \Gamma$	
$a, \varepsilon; c$ when $a \in \Sigma, c \in \Gamma$	
$a, b; \varepsilon$ when $a \in \Sigma, b \in \Gamma$	
$a, \varepsilon; \varepsilon$ when $a \in \Sigma$	

How does the meaning change if  $a$  is replaced by  $\varepsilon$ ?

*Note: alternate notation is to replace ; with  $\rightarrow$*

## Week4 friday

For the PDA state diagrams below,  $\Sigma = \{0, 1\}$ .

Mathematical description of language

State diagram of PDA recognizing language  
 $\Gamma = \{\$, \#\}$



$\Gamma = \{ @, 1 \}$



$$\{0^i 1^j 0^k \mid i, j, k \geq 0\}$$

*Big picture:* PDAs were motivated by wanting to add some memory of unbounded size to NFA. How do we accomplish a similar enhancement of regular expressions to get a syntactic model that is more expressive?

DFA, NFA, PDA: Machines process one input string at a time; the computation of a machine on its input string reads the input from left to right.

Regular expressions: Syntactic descriptions of all strings that match a particular pattern; the language described by a regular expression is built up recursively according to the expression's syntax

**Context-free grammars:** Rules to produce one string at a time, adding characters from the middle, beginning, or end of the final string as the derivation proceeds.

Term	Typical symbol	Definition
<b>Context-free grammar</b> (CFG)	$G$	$G = (V, \Sigma, R, S)$
<b>Variables</b>	$V$	Finite set of symbols that represent phases in production pattern
<b>Terminals</b>	$\Sigma$	Alphabet of symbols of strings generated by CFG $V \cap \Sigma = \emptyset$
<b>Rules</b>	$R$	Each rule is $A \rightarrow u$ with $A \in V$ and $u \in (V \cup \Sigma)^*$
Start variable	$S$	Usually on LHS of first / topmost rule
<b>Derivation</b>	$S \Rightarrow \dots \Rightarrow w$	Sequence of substitutions in a CFG Start with start variable, apply one rule to one occurrence of a variable at a time
<b>Language</b> generated by the CFG $G$	$L(G)$	$\{w \in \Sigma^* \mid \text{there is derivation in } G \text{ that ends in } w\} = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$
<b>Context-free language</b>		A language that is the language generated by some CFG
Sipser pages 102-103		

**Examples of context-free grammars, derivations in those grammars, and the languages generated by those grammars**

$G_1 = (\{S\}, \{0\}, R, S)$  with rules

$$S \rightarrow 0S$$

$$S \rightarrow 0$$

In  $L(G_1)$  ...

Not in  $L(G_1)$  ...

$$G_2 = (\{S\}, \{0, 1\}, R, S)$$

$$S \rightarrow 0S \mid 1S \mid \varepsilon$$

In  $L(G_2) \dots$

Not in  $L(G_2) \dots$

$(\{S, T\}, \{0, 1\}, R, S)$  with rules

$$S \rightarrow T1T1T1T$$

$$T \rightarrow 0T \mid 1T \mid \varepsilon$$

In  $L(G_3) \dots$

Not in  $L(G_3) \dots$

$G_4 = (\{A, B\}, \{0, 1\}, R, A)$  with rules

$$A \rightarrow 0A0 \mid 0A1 \mid 1A0 \mid 1A1 \mid 1$$

In  $L(G_4)$  ...

Not in  $L(G_4)$  ...

*Extra practice:* Is there a CFG  $G$  with  $L(G) = \emptyset$ ?