CSCI 361 Lecture 7: Context-Free Grammars

Shikha Singh

Announcements & Logistics

- **HW 3** was due last night
- **HW 4** will be released today and due next Wed (Oct 1)
- HW 2 graded feedback released
  - Let me know if you have any questions
- Solutions to HW 1 and HW 2 are on GLOW
- Hand in **Exercise # 6** and pick up **Exercise # 7**
- **Reminder:** Midterm 1 in-class on Oct 7
  - Everything up to HW 4 included
- Practice Midterm will be released Oct 1
- **Question**. Can you view the course calendar on the webpage?

Last Time

- Pumping lemma to prove languages are not regular
- More practice with recognizing non-regular languages and proving non-regularity

Today

- New model of computation that is slightly more powerful
  - Context-free languages

Last Time: Use Pumping Lemma

Problem 1. Prove that the language $L = \{a^i b^i \mid i \in \mathbb{N}\}$ is not regular.

Problem 2. Prove that $L = \{ww^R \mid w \in \{0,1\}^*\}$ is not regular.

Problem 3. Is the language $L = \{(ab)^i \circ (ab)^i \mid i \geq 0\}$ regular?

Problem 4. Prove that

$L = \{w \mid w \in \{0,1\}^*$ and the number of 1s in w is not equal to the number of 0s in $w\}$

is not regular.

Solutions : Use Pumping Lemma

Problem 1. Prove that the language $L = \{a^i b^i \mid i \in \mathbb{N}\}$ is not regular.

Problem 2. Prove that $L = \{ww^R \mid w \in \{0,1\}^*\}$ is not regular.

Problem 1 and 2 solutions in the textbook.

Problem 3. Is the language $L = \{(ab)^i \circ (ab)^i \mid i \geq 0\}$ regular?

Yes! Can draw a DFA or regular expression $(abab)^*$

Problem 4. Prove $L = \{w \mid w \in \{0,1\}^* w$ has unequal number of 0s and 1s$\}$ is not regular.

Suppose L is regular, then $\overline{L}$ should be regular (closed under intersection). But,
$\overline{L} = \{w \in \{0,1\}^* \mid w$ *has equal number of 0s and 1s*$\}$. *We proved $\overline{L}$ is not regular!*

Finite Automata Limitations

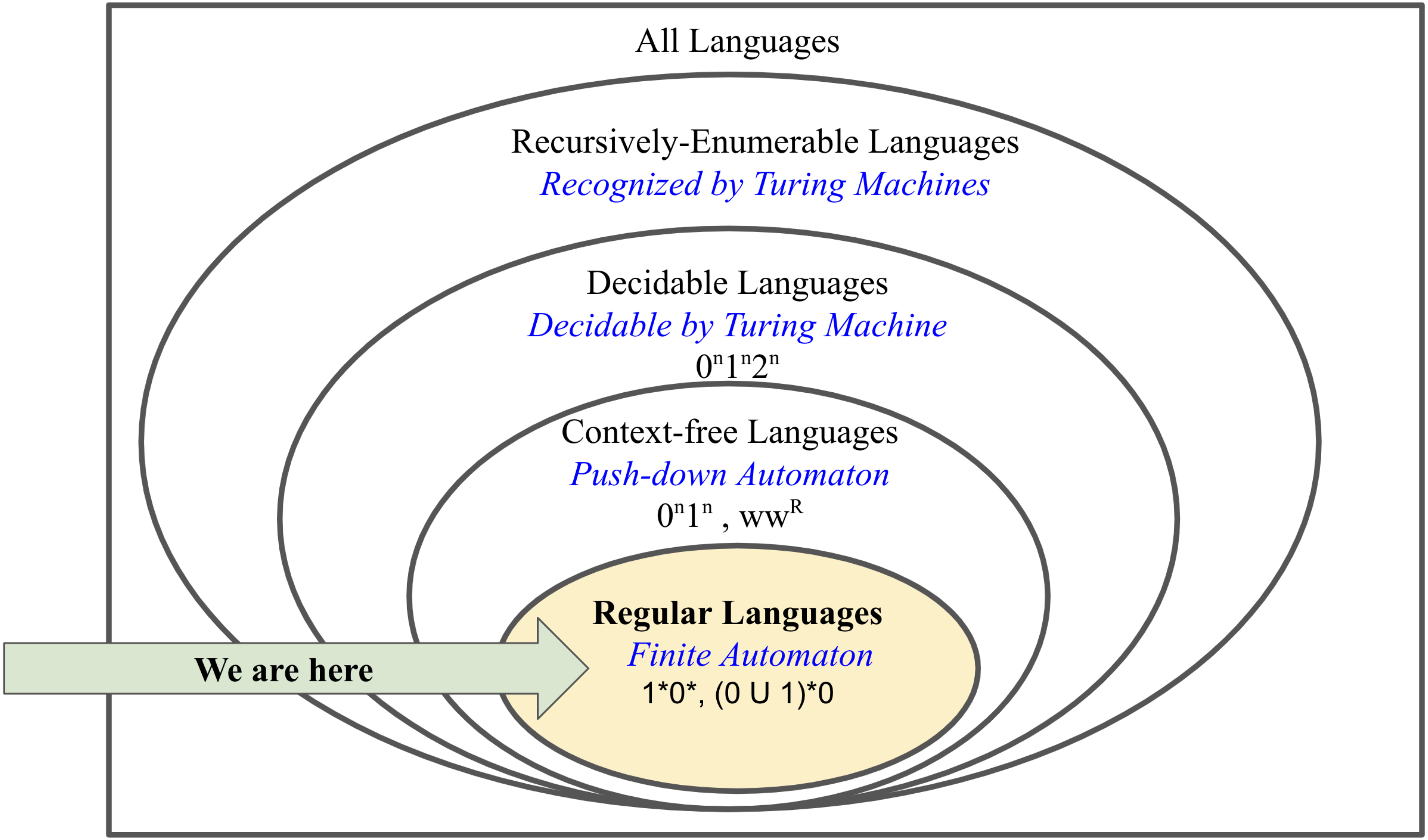
- Code snippets that can only store finitely many states
- (Theory vs PL) Remember that all models of computation we discuss map directly to programming constructs

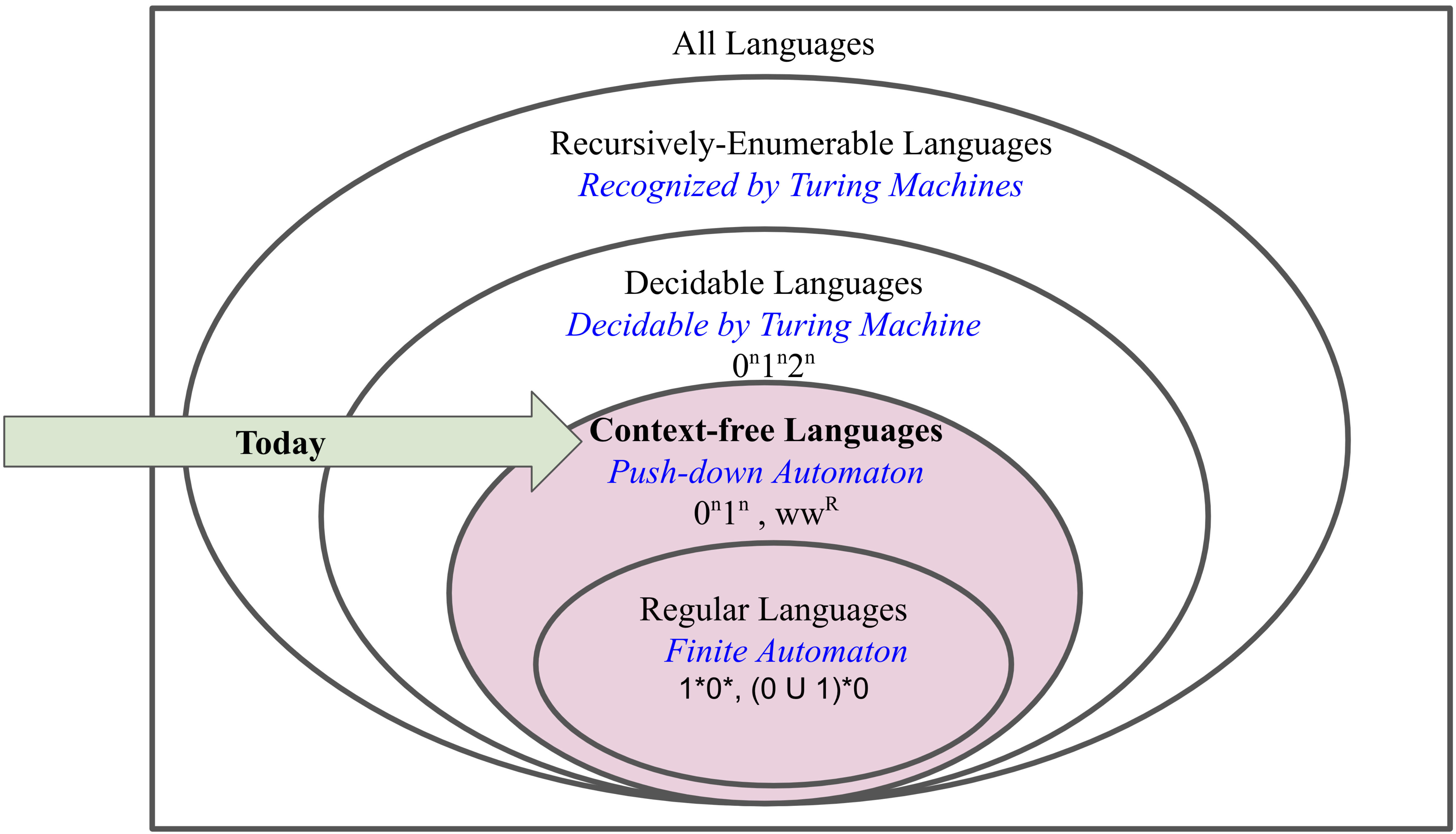
```
CONTAINS11(w[1..n]):
  found ← FALSE
  for i ← 1 to n
      if i = 1
          last2 ← w[1]
      else
          last2 ← w[i − 1] · w[i]
      if last2 = 11
          found ← TRUE
  return found
```

```
MULTIPLEOF5(w[1..n]):
  rem ← 0
  for i ← 1 to n
      rem ← (2 · rem + w[i]) mod 5
  if rem = 0
      return TRUE
  else
      return FALSE
```

Finite Automata Applications

- Lexical analysis and parsing in compilers and programming in general
- Networking protocols and routing
- Circuit design and event-driven programming
- Synchronization of distributed devices


All Languages
Recursively-Enumerable Languages
Recognized by Turing Machines
Decidable Languages
Decidable by Turing Machine
$0^n1^n2^n$
Context-free Languages
Push-down Automaton
0^n1^n , ww^R
Regular Languages
Finite Automaton
1*0*, (0 U 1)*0
We are here


All Languages
Recursively-Enumerable Languages
Recognized by Turing Machines
Decidable Languages
Decidable by Turing Machine
$0^n1^n2^n$
Today
Context-free Languages
Push-down Automaton
0^n1^n , ww^R
Regular Languages
Finite Automaton
1*0*, (0 U 1)*0

Context-Free Grammar

- Generative model to specify the next class of languages
- First used in the study of natural/human languages
- Applications in specification & compilation of programming languages
    - Syntax of a PL can be specified using its grammar
    - Compiler to check correct syntax uses a parser to check against valid rules

Example CFG

- CFGs consists of a collection of substitution rules, called productions
- Left-hand side of a rule has a **single variable** (or **non-terminal**)
- Right-hand side can consist of variables and **terminals**
- ***Conventions***: upper-case letters for variables/non-terminals, lower-case letters for terminals,
    - S for start variable, usually on the LHS of the topmost rule
- Example: $S \to 0\ S\ 1$

  $S \to \varepsilon$

Derivations to Generarte Strings

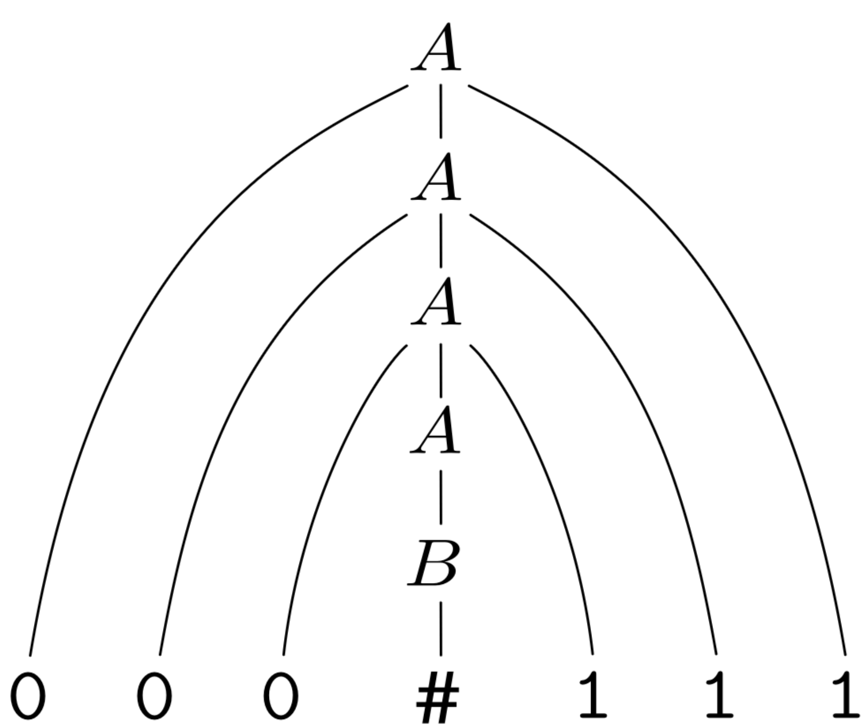
- A sequence of substitutions starting with the start variable and ending in a string of terminals is a ***derivation***
- For example, the derivation of 000111 using the grammar
  $S \rightarrow 0\ S\ 1$
  $S \rightarrow \varepsilon$
- $S \implies 0S1 \implies 00S11 \implies 000S111 \implies 000111$
- Can you guess the language of this grammar?
  - $L = \{0^n 1^n \mid n \geq 0\}$
- Thus, CFGs are more powerful than RegExp/DFA/NFAs

Language of a Grammar

- The set of all strings that can be generated using the rules of a grammar constitute the language of the grammar
- Any language that can be generated by some context-free grammar is called a **context-free language**

Parse Trees

- Rooted trees that represent a derivation
  - Root: start variable, leaves: derived string
  - Children of nodes represent the rule that is being applied
- Will be useful in discussing context-free languages



$$A \to 0\ A\ 1$$
$$A \to B$$
$$B \to \#$$

Formal Definition: CFG

- A context-free grammar G is a quadruple (V, Σ, R, S) where
  - V is a finite set called **variables**
  - Σ is a finite set (disjoint from V) called the **terminals**
  - R is a finite subset of $V \times (V \cup \Sigma)^*$ called rules, and
  - S (the start symbols) is a element of V
- For any $A \in V$ and $u \in (V \cup \Sigma)^*$, we write $A \rightarrow u$ if $(A, u) \in R$

Language of a Grammar

- If $v, w, v \in (V \cup \Sigma^*)$ and $A \rightarrow w$ is a rule, then we say uAv **yields** uwv and write $uAv \implies uwv$
- We say u **derives** v denoted $u \overset{*}{\implies} v$, if there exists a sequence $u_1, \ldots, u_k$ such that

$$u \implies u_1 \implies \cdots u_k \implies v$$

- The language of the grammar G is $L(G) = \{w \mid S \overset{*}{\implies} w\}$

Grammar for English

A grammar for the English language tells us whether a sentence is "well formed". For example:

<Sentence> → <NounPhrase><VerbPhrase>
<NounPhrase> → <Article><NounUnit>
<NounUnit> → <Noun> | <Adjective><NounUnit>
<VerbPhrase> → <Verb> <NounPhrase>
<Article> → a | the
<Adjective> → big | small | black | green | colorless
<Noun> → dog | cat | mouse | bug | ideas
<Verb> → loves | chases | eats | sleeps

Some generated sentences:
The black dog loves the small cat
A cat chases a mouse
The colorless bug chases the green ideas

Example: Programming Language Syntax

<program> → <block>
<block> → { <command-list> }
<command-list> → ε
<command-list> → <command> <command-list>
<command> → <block>
<command> → <assignment>
<command> → <one-armed-conditional>
<command> → <two-armed-conditional>
<command> → <while-loop>
<assignment> → <var> := <expr>
<one-armed-conditional> → **if** <expr> <command>
<two-armed-conditional> → **if** <expr> <command> **else** <command>
<while-loop> → **while** <expr> <command>

Possible generated program

```
{ x := 4
  while x >1
      x := x -1 }
```

Parsing

- A compiler for a programming language takes an input program in the language and converts it to a form more suitable for execution
- To do so, the compiler creates a parse tree of the code to be compiled using its CFG: this process is called parsing

Regular Languages are Context-Free

- Every regular language can be described by some CFG
- **Takeaway:** CFGs are more "expressive" in power than regular expressions

Regular Languages are Context-Free

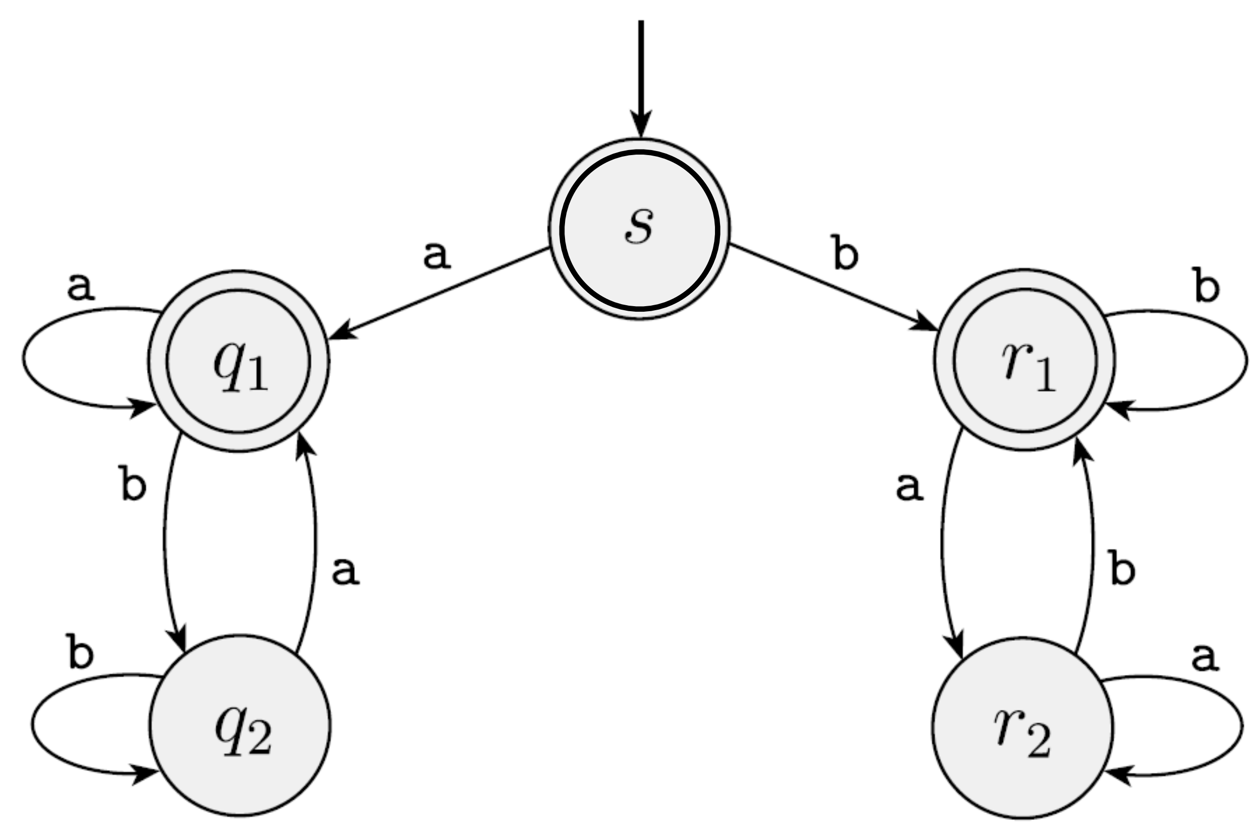
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA for the regular language L
- We can construct a CFG G for L as follows
  - Make a variable Q_i for each state $q_i \in Q$
  - For each $q_i, q_j \in Q$ and $a \in \Sigma$ such that $\delta(q_i, a) = q_j$ a rule $Q_i \rightarrow a\ Q_j$ add a rule $Q_i \rightarrow a\ Q_j$
  - Make Q_0 the start variable
  - Add $Q_i \rightarrow \varepsilon$ if $q_i \in F$

Regular Languages are Context-Free

- Proof of correctness: $L(M) = L(G)$
- Suppose M accepts w, then there exists a sequence of states $q_0, q_1, \ldots, q_n$ that M enters when reading $w_1, \ldots, w_n$ st $q_n \in F$
- There exists derivation $Q_0 \implies w_1 Q_1 \implies \cdots \implies w_n Q_n$ in G and since $q_n \in F$, we have the rule $Q_n \to \varepsilon$, thus $w \in L(G)$
- The other direction is analogous

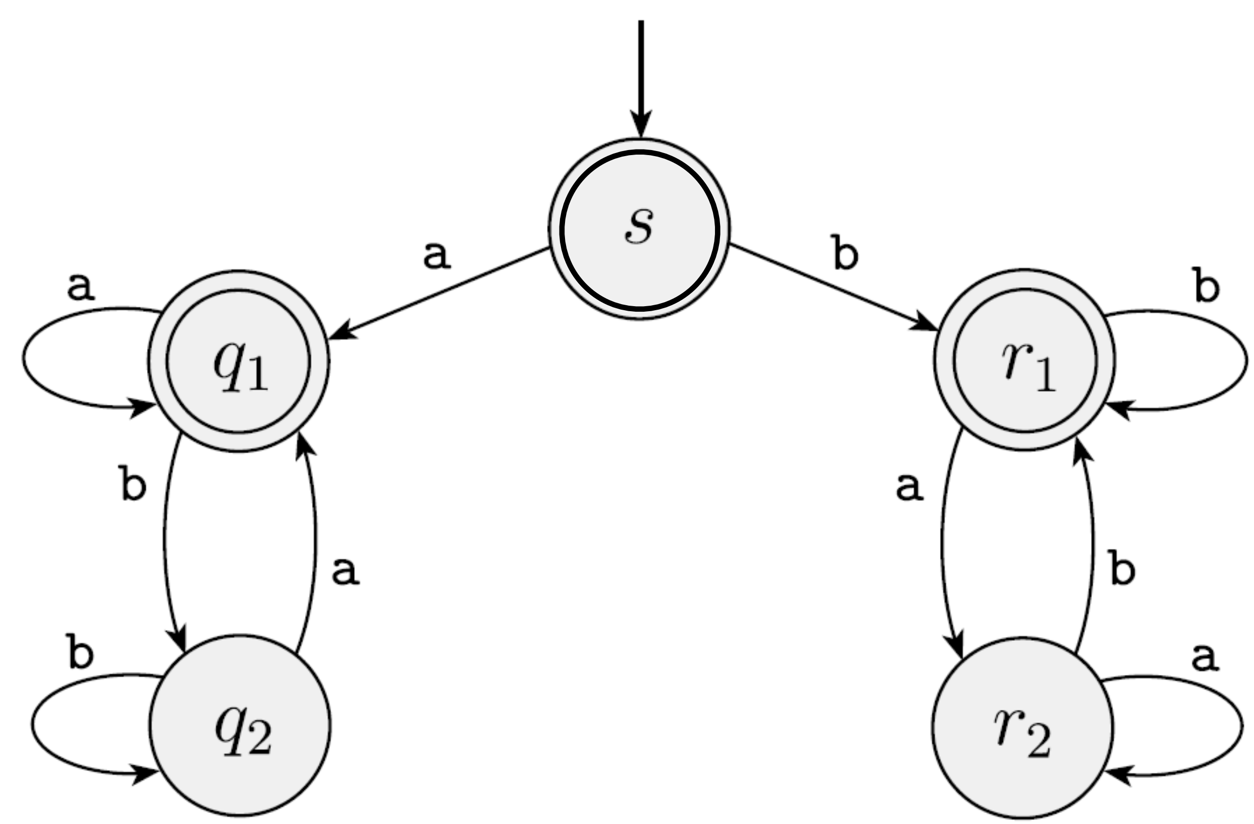
Regular Languages are Context-Free

- Direct translation?



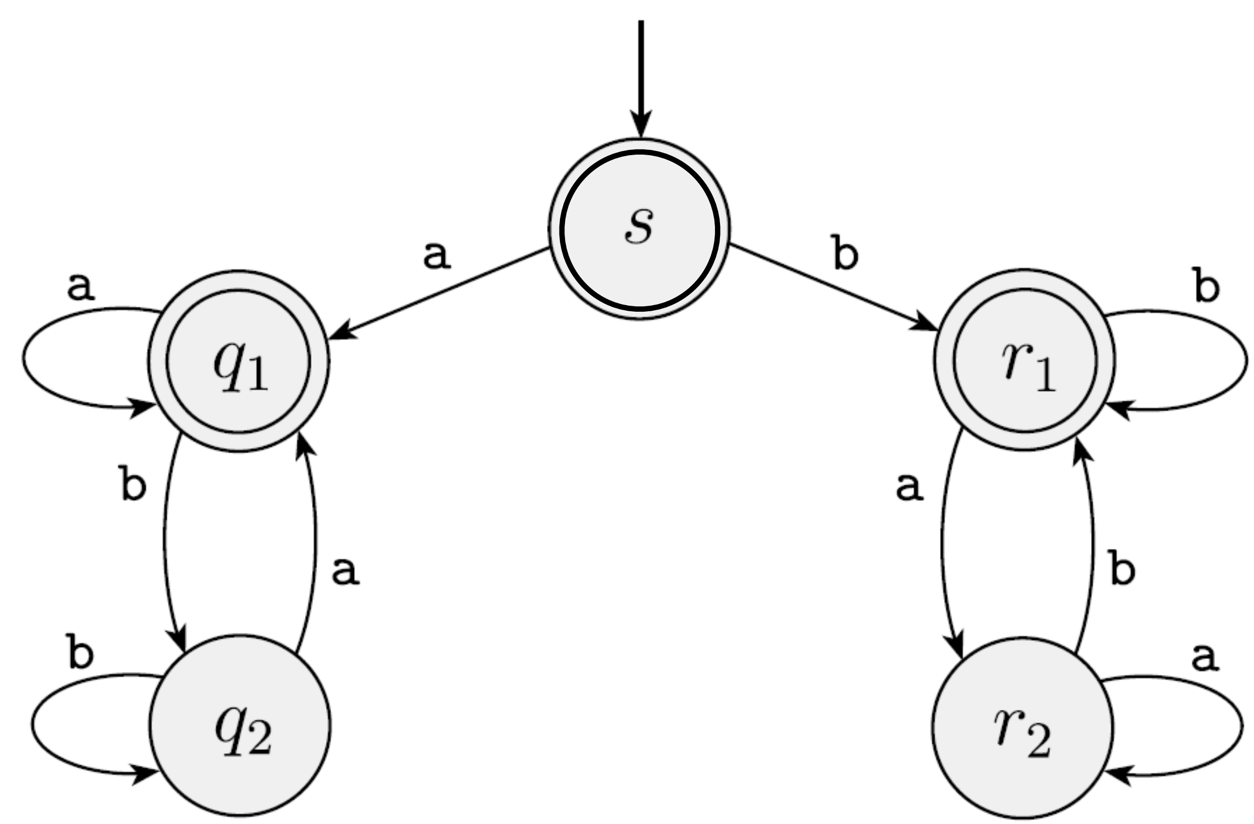
Regular Languages are Context-Free

- $S \rightarrow aQ_1 \mid bR_1 \mid \varepsilon$
- $Q_1 \rightarrow aQ_1 \mid bQ_2$
- $Q_2 \rightarrow aQ_1 \mid bQ_2$
- $R_1 \rightarrow bR_1 \mid aR_2$
- $R_2 \rightarrow bR_1 \mid aR_2$
- Can create an easier CFG by breaking down into small pieces



Regular Languages are Context-Free

- Union of strings that start and end in a, start and end in b, and ε
- $S \rightarrow A \mid B \mid \varepsilon$
- $A \rightarrow aTa$
- $B \rightarrow bTb$
- $T \rightarrow aT \mid bT \mid \varepsilon$ (generates Σ^*)
- More intuitive!



Regular Grammars

- A CFG is **regular** if any occurrence of a variable on the RHS of a rule is as the rightmost symbol
- If a CFG is regular, there is a DFA that recognizes the same language
  - $Q = V \cup \{f\}$ (A state for each variable plus an accept state)
  - Rule $A \rightarrow aB$ becomes $\delta(A, a) = B$
  - If there is a $A \rightarrow a$ then $\delta(A, a) = f$

Exercise: Practice with CFGs

Describe a CFG for the following languages

- $L = \{w \in \{a, b\}^* \mid |w|$ is even $\}$
- $L = \{w \in \{0,1\}^* \mid w = w^R\}$
- $L = \{w \in \{a, b\}^* \mid w$ has the same # of a's and b's$\}$

Solutions of CFGs

- $L = \{w \in \{a, b\}^* \mid |w| \text{ is even } \}$

  $S \rightarrow aT \mid bT \mid \varepsilon$

  $T \rightarrow aS \mid bS$

- $L = \{w \in \{a, b\}^* \mid w = w^R\}$

  $S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$

- $L = \{w \in \{a, b\}^* \mid w \text{ has the same \# of a's and b's}\}$

  $S \rightarrow SS$

  $S \rightarrow aSb$

  $S \rightarrow bSa$

  $S \rightarrow \varepsilon$

Correctness Proof: Induction

$S \rightarrow SS$

$S \rightarrow aSb$

$S \rightarrow bSa$

$S \rightarrow \varepsilon$

To prove: $L(G) = \{w \mid w$ has an equal # of a's and b's$\}$

($\Longrightarrow$) Consider any $w \in L(G)$ and induct on the length k of derivation of w

(a) $k = 1$ then $S \implies \varepsilon$ and ε has equal # of a's and b's

(b) $k > 1$ then either $S \implies SS \overset{*}{\implies} xy$

or $S \implies aSb \overset{*}{\implies} axb$

or $S \implies bSa \overset{*}{\implies} aya$

In each case, S derives x, y in less than k steps and by IH, they must have equal number of a's and b's

Correctness Proof: Induction

To prove: $L(G) = \{w \mid w$ has an equal # of a's and b's$\}$

$S \to SS$

$S \to aSb$

$S \to bSa$

$S \to \varepsilon$

($\Leftarrow$) Consider any w with equal # of a's and b's

Can show $w \in L(G)$ by induction on $|w|$

(a) $|w| = 0$ then $w = \varepsilon$

(b) $|w| = k + 2$ (as $|w|$ must be even)

Can divide by 4 cases depending on first and last symbol of w, in each case show that the smaller string can be derived by IH

Case (i) and (ii) $w = axb$ or $w = bxa$

Case (iii) and (iv) $w = axa$ and $w = bxb$

CFG for this Language?

- CFG for $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$
- Union of $L_1 = \{a^i b^i c^j \mid i, j \geq 0\}$ and $L_2 = \{a^i b^j c^j \mid i, j \geq 0\}$

Closure Properties of CFLs

- CFLs are closed under
  - Union
  - Concatenation
  - Kleene star
- **Important**. Not closer under complement and intersection!

Closure Properties of CFLs

Given $G_1 = (V_1, \Sigma_1, R_1, S_1)$
$G_2 = (V_2, \Sigma_2, R_2, S_2)$

Union: $L(G_1) \cup L(G_2)$ is generated by
$R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\}$

NB: Assume that $V_1 - \Sigma_1, V_2 - \Sigma_2$ are disjoint.

Closure Properties of CFLs

Given $G_1 = (V_1, \Sigma_1, R_1, S_1)$
$G_2 = (V_2, \Sigma_2, R_2, S_2)$

Union: $L(G_1) \cup L(G_2)$ is generated by
$R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\}$

Concatenation: $L(G_1)L(G_2)$ is generated by
$R_1 \cup R_2 \cup \{S \to S_1S_2\}$

NB: Assume that $V_1 - \Sigma_1, V_2 - \Sigma_2$ are disjoint.

Closure Properties of CFLs

Given $G_1 = (V_1, \Sigma_1, R_1, S_1)$

$G_2 = (V_2, \Sigma_2, R_2, S_2)$

Union: $L(G_1) \cup L(G_2)$ is generated by

$R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\}$

Concatenation: $L(G_1)L(G_2)$ is generated by

$R_1 \cup R_2 \cup \{S \to S_1 S_2\}$

Kleene $*$: $L(G_1)^*$ is generated by

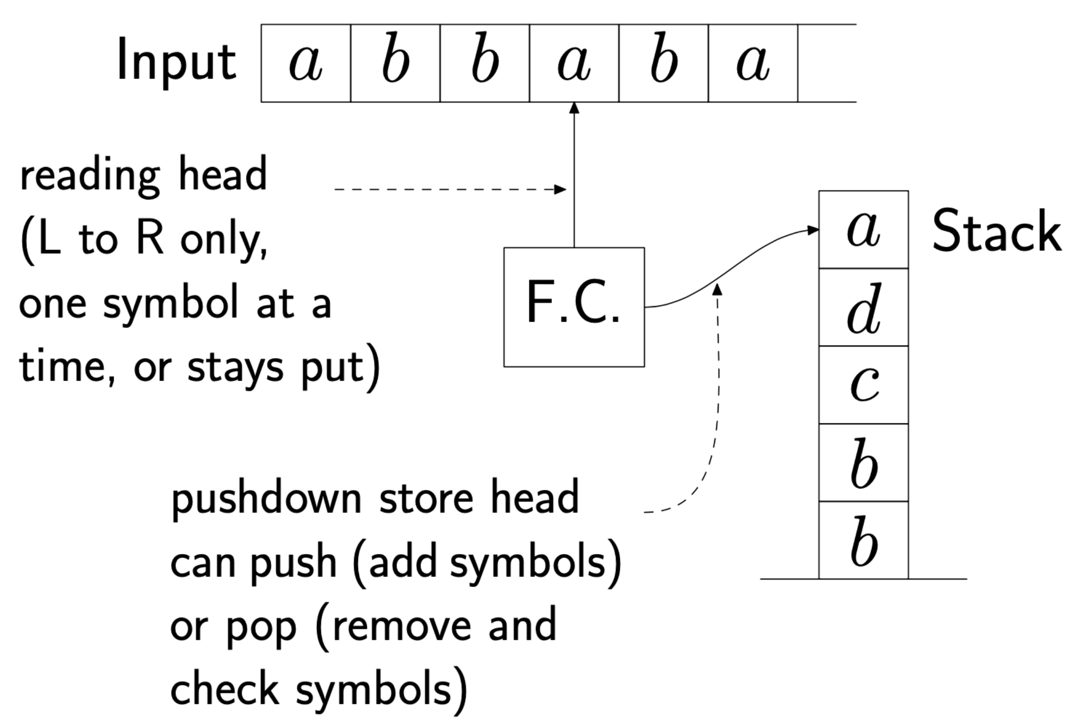
$R_1 \cup \{S \to e | S \to S_1 S\}$

Automata for CFGs

- Regular Languages : Finite Automata
- Context-free languages: ??

Pushdown Automata

- Basically an NFA with a stack (pushdown store)
- The stack can consist of unlimited number symbols but can only be read and altered at the top:
    - Can only pop symbol from top or push symbol to top



Pushdown Automata Transitions

- Transitions of a PDA have two parts:
  - **State transition** and **stack manipulation** (push/pop)
  - If in state p reading input symbol a and b on the stack, replace b with c on the stack and enter state q
    - $(p, a, b) \to (q, c)$
    - $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathscr{P}(Q \times \Gamma_\varepsilon)$
  - In state diagram arrow goes from $p \to q$ with label $a, b \to c$

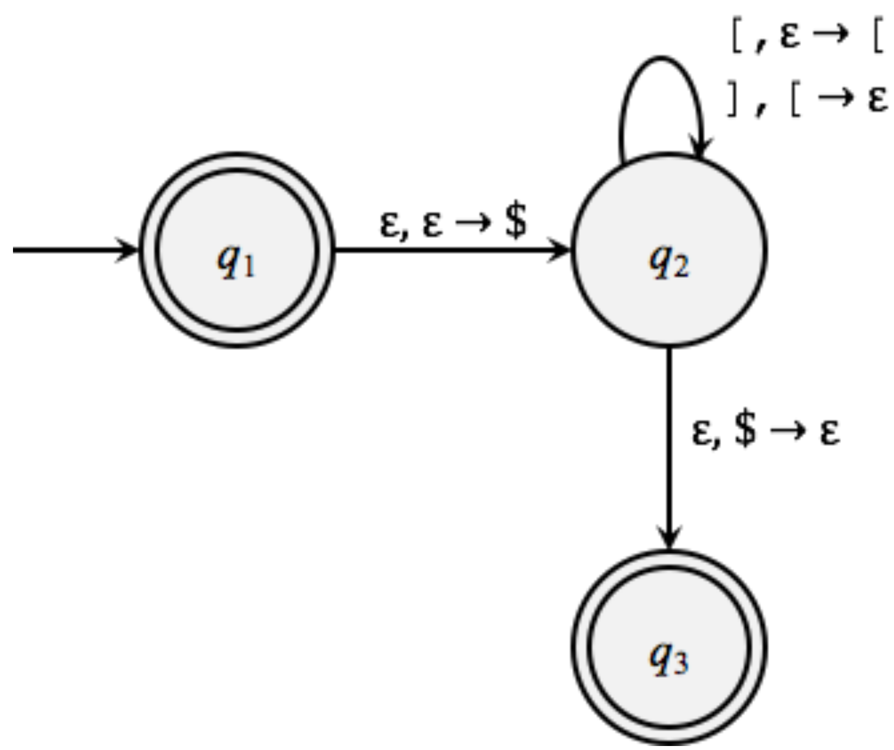
Formal Definition: PDA

- A pushdown automaton is a six tuple $M = (Q, \Sigma, \Gamma, \delta, q_0 F)$ where
  - Q is the finite set of states
  - Σ is a finite alphabet (the input symbols)
  - Γ is a finite tape alphabet (the stack symbols)
  - $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathscr{P}(Q \times \Gamma_\varepsilon)$ is the transition function
  - $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of accept states

Example PDA

- Consider the language over $\Sigma = \{[,]\}$ of all strings made up of correctly nested brackets
- CFG for this language: $S \rightarrow \varepsilon \mid [S] \mid SS$
- Now lets create a push-down automata for this language
- What to store on the stack?

Example PDA for Balanced Brackets



Recall: A transition of the form a, b → z means "if the current input symbol is a and the current stack symbol is b, then follow this transition, pop b, and push the string z"