CSCI 361 Lecture 8: Context-Free Grammars

Shikha Singh

Announcements & Logistics

- **HW 3** out, due Oct 9 (next Wed)
    - Slightly longer: 6 questions but with subparts
    - Recommend finishing Q1-3 by this Wed, 4-6 next week
    - Happy to provide feedback on write up so you can revise solutions
- Hand in **reading questions # 5** and pick up **reading questions #6**
- What I did last summer colloquium this Friday
    - Sign up if you would like to present
    - https://forms.gle/Krg1f71gkU7qTpHe9

Last Time

- Discussed alternate tools for proving languages are not regular
    - Pumping lemma
    - Closure properties
- Important to know how to use both approaches: Myhill-Nerode and PL
    - Depending on the language, one might be easier than other

Review PL Steps

- Proving L is not regular using pumping lemma
  - Assume L is regular, let p be the pumping lemma given by lemma
  - Consider a specific string $w \in L$ of length at least p such that
  - for every possible partition of w into x, y, z satisfying
    - $|xp| \leq p$ and $|y| > 0$
  - there exists an i such that $xy^iz \notin L$
- The above steps provide a contradiction to L being regular by PL
- HW 4 Problem 5
  - Show that a language is not regular and show that it satisfies conditions of the pumping lemma

Leftovers: Regular or Not

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

Leftovers: Closure Question

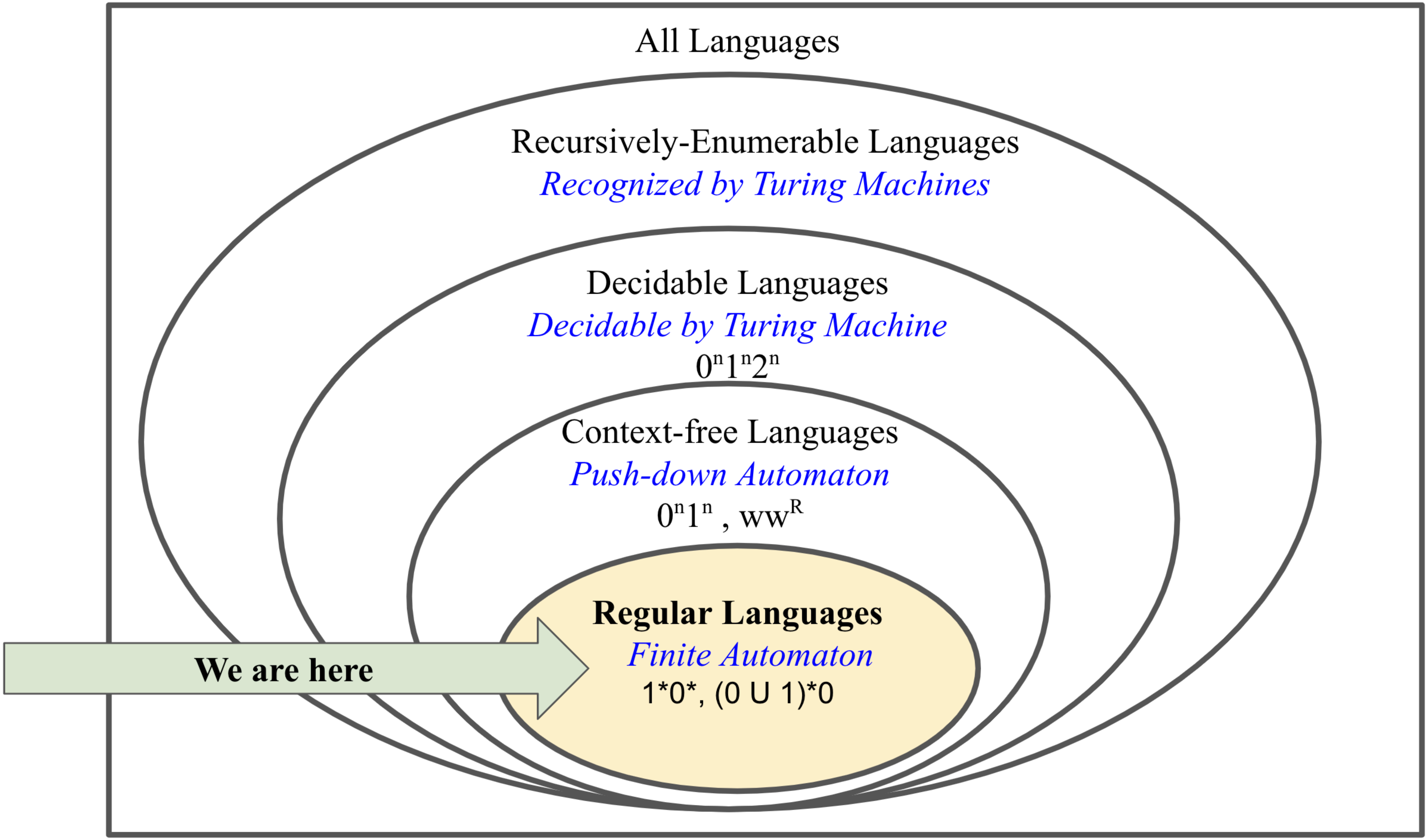
Question. Are all subsets of regular languages also regular?

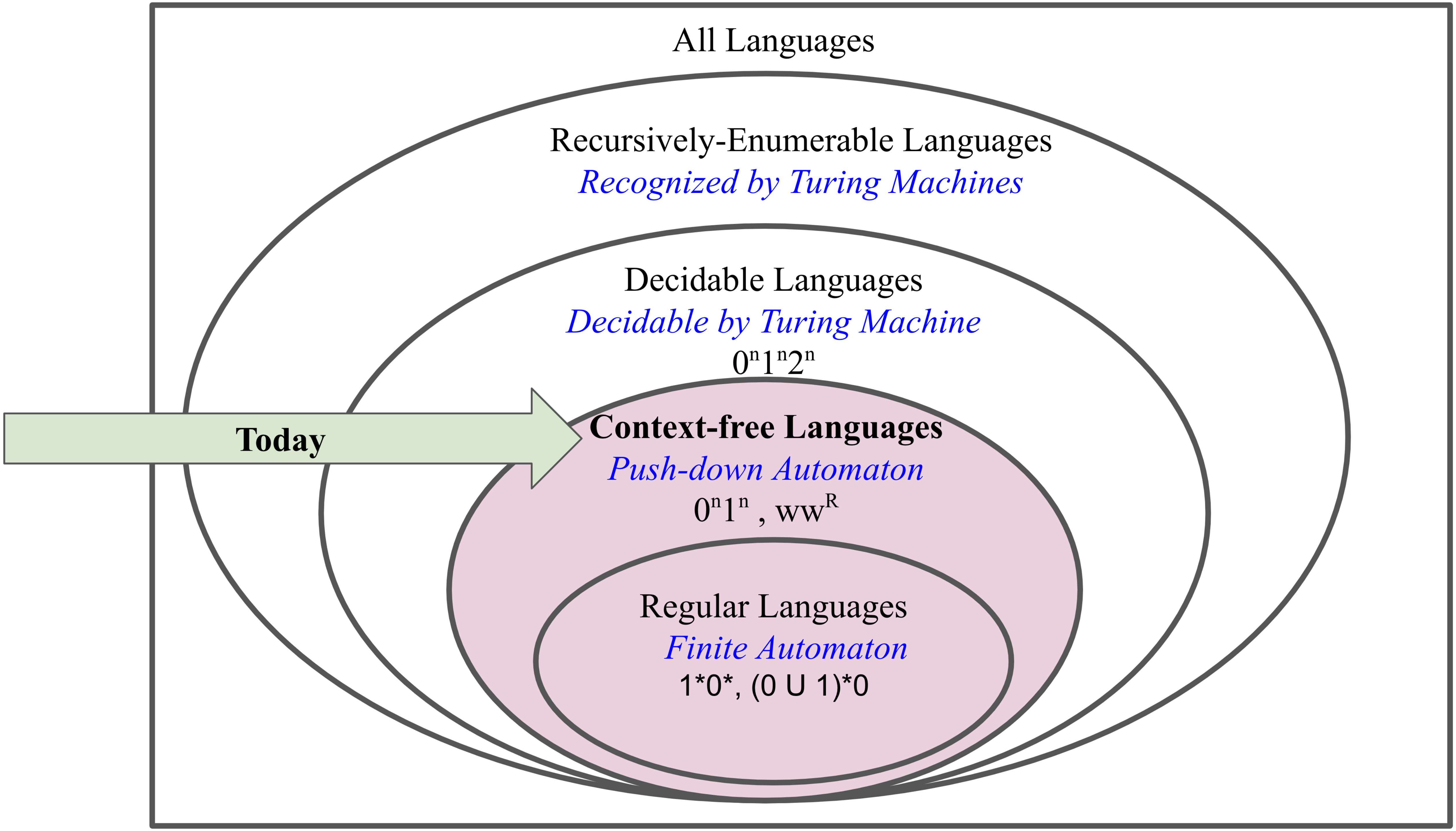
Finite Automata Applications

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

Firing Squad Problem

- Cellular automata: finite automata where each cell changes state based on current state and state of neighbors
- https://www.youtube.com/watch?v=xV1aKUdlIjU


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 regular exp/DFA/NFAs

Language of a Grammar

- 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

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 $L(G) = \{w \mid S \overset{*}{\implies} w\}$

Examples of CFGs

Describe a CFG for the following languages

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

Solutions of CFGs

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

$S \rightarrow SS$

$S \rightarrow aSb$

$S \rightarrow bSa$

$S \rightarrow \varepsilon$

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

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

$T \rightarrow aS \mid bS$

- $L = \{w \in \{0,1\}^* \mid w = w^R\}$

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

Correctness Proof: Induction

$S \to SS$

$S \to aSb$

$S \to bSa$

$S \to \varepsilon$

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 $S \Longrightarrow \varepsilon$ and ε has equal # of a's and b's

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

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

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

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

$S \rightarrow SS$

$S \rightarrow aSb$

$S \rightarrow bSa$

$S \rightarrow \varepsilon$

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

($\Leftarrow$) 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 $w = bxb$

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

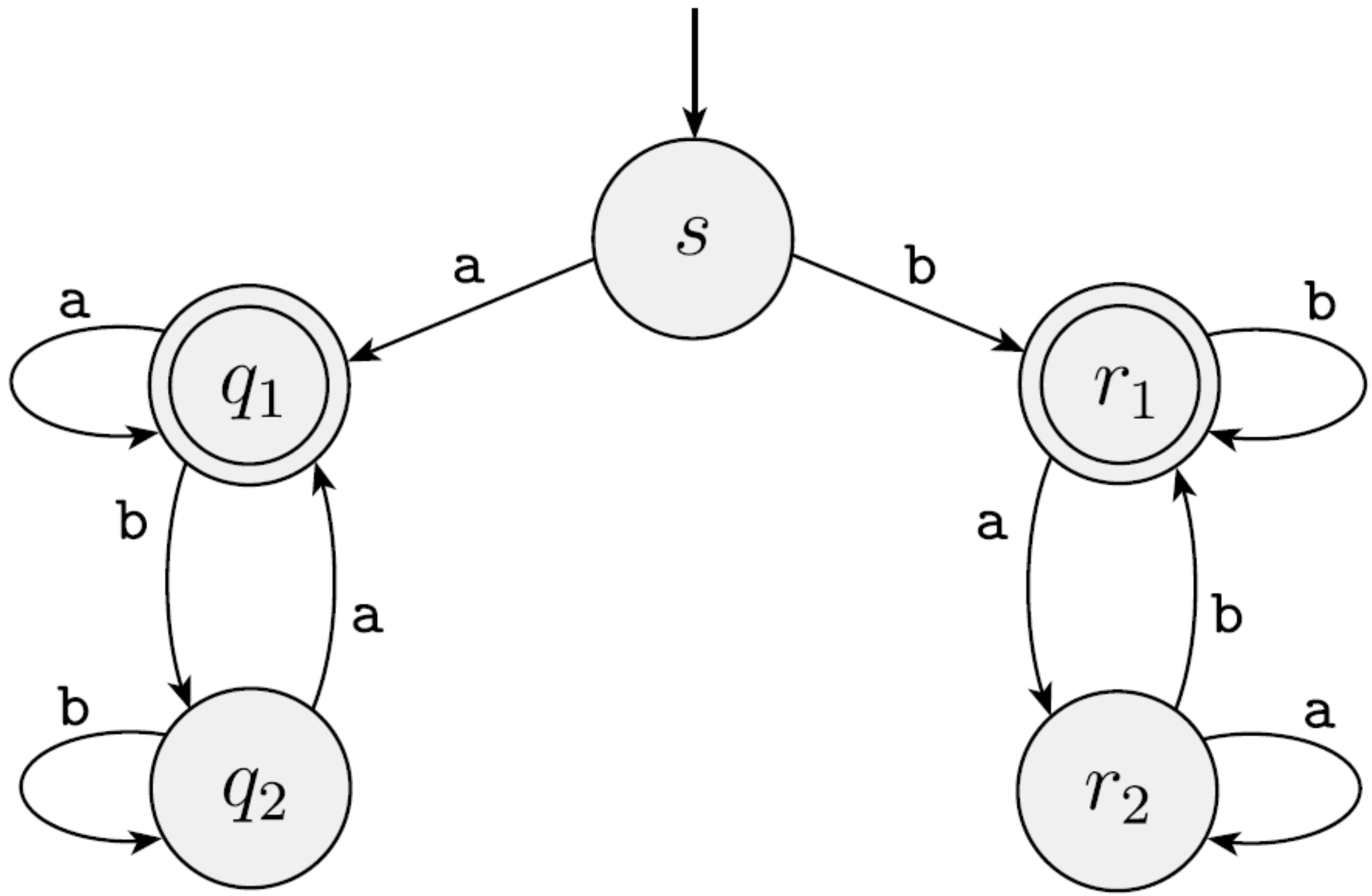
Regular Languages are Context-Free

- 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 $Q_i \to a\ Q_j$ add a rule $Q_i \to a\ Q_j$
  - Make Q_0 the start variable
  - Add $Q_i \to \varepsilon$ if $q_i \in F$

Regular Languages are Context-Free



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 $A \to aB$ becomes $\delta(A, a) = B$
  - If there is a $A \to a$ then $\delta(A, a) = f$

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