

- Optional Exam #1: Wednesday October 29, 9:00-10:00
- Come to the lecture room a bit early, to find a seat
- Exam questions come from exercise sheets #1 and #2
- Reading: Chapters 1 and 2
- Last time: regular languages, non-regular languages, Pumping Lemma
- Today: non-regular languages, Pumping Lemma, grammars

The Pumping Lemma (PL) for Regular Languages

Theorem

(PL) If A is a regular language, then there exists a number p (the pumping length) so that if s is any string in A of length at least p , then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

- 1 $xy^iz \in A$, for each $i \geq 0$,
- 2 $|y| > 0$, and
- 3 $|xy| \leq p$.

Proving a Language Non-Regular

- To prove a language is regular: build a FA or give a RE
- To prove a language is non-regular, we use the PL

Claim

$L = \{0^n 1^n \mid n \geq 0\}$ is not a regular language

Proof.

- Suppose L is regular. Then the PL applies.
- Let p be the pumping length.
- Let's pick a string $s = 0^p 1^p$ from L and longer than p .
- Now we must consider all possible ways to break s into three parts $s = xyz$ subject to PL conditions $|y| > 0$ and $|xy| \leq p$.
- Clearly, y must contain one or more zeros (from conditions 2 and 3).
- Condition 1 of the PL requires that $xy^i z \in L$ for each $i \geq 0$.
- But this is not possible: "pumping" y up or down results in strings that are not in L , $xz \notin L$ (fewer 0s than 1s), $xyyz \notin L$ (more 0s than 1s).

Proving a Language Non-Regular



$$s = 00^z 0100^z 01$$

$$0^z, 1, 00, 01, 100, 000, \dots$$

$$x \ x \ v \ x \ x \ v$$

Claim

$L = \{ww \mid w \in \{0,1\}^*\}$ is not a regular language

Proof.

Suppose L is regular. Then the PL applies. Let p be the pumping length. Then consider the string $s = 0^p 1 0^p 1$ and all possible ways to break s down into $s = xyz$ subject to the 3 PL conditions. Because of condition 3, y must contain one or more 0s. Then xy^2z has more 0s in on the left side of the first 1 than on the right, which means that $xy^2z \notin L$. □

Again note how the careful choice of s reduces the number of cases we need to consider.

some choices
don't work e.g. $0^p 0^p$

Proving a Language Non-Regular

0, 1, 00, 01, 10, 11, 000, 001, 010, 011, ...
✓ ✗ ✓ ✗ ✗ ✗ ✓ ✗ ✗

Claim

$L = \{0^i 1^j \mid i > j\}$ is not a regular language

$(0 \neq y^i z \in L \quad \forall i = 0, 1, 2, \dots)$

Proof.

Suppose L is regular. Then the PL applies. Let p be the pumping length. Then consider the string $s = 0^{p+1}1^p$ and all possible ways to break s down into $s = xyz$ subject to the 3 PL conditions.

Because of condition 3, y must contain one or more 0s.

What can we say about xy^2z and its membership in L ?

Better to consider $xy^0z = xz$, as this string has less than or equal number of 0s than 1s, which means that $xy^0z \notin L$. □

Note that here “pumping up” didn’t work, but “pumping down” does indeed work.

The Pumping Lemma (PL) for Regular Languages

Observe

- the PL is not a necessary and sufficient condition for regularity!
- that is, the pumping lemma is not a complete characterization of regular languages.
- PL: $A \text{ regular} \Rightarrow p : s \in A \text{ and } |s| > p \dots$
- The opposite is not necessarily true

Context Free Languages

CFL
RL

- Regular languages are characterized by FAs and REs
- CFLs are characterized by pushdown automata (PDAs) and Context Free Grammars (CFGs)
- Example of a CFL: $L = \{0^n 1^n | n \geq 0\}$
- PDAs are machines with memory (stack)
- Unlike FAs, DPDAs and NPDAs are not equivalent in power
- CFGs are Type 2 grammars in Chomsky's hierarchy

Type 2 ~ CFL
Type 3 ~ RL

Context Free Grammars

Definition

A context-free grammar is a 4-tuple (V, Σ, R, S) , where:

- 1 V is a finite set of variables,
- 2 Σ is a finite set of terminals, $\Sigma \cap V = \emptyset$
- 3 R is a finite set of rules, $\alpha \rightarrow \beta$, where α is a variable and β a string of variables and terminals
- 4 $S \in V$ is the start variable

Example: $G = (\{S\}, \{0, 1\}, R, S)$, where the set of rules R is given

by: $S \rightarrow 0S1 \mid \epsilon$

Handwritten derivation tree for the string 000S111:
 $S \rightarrow 0S1 \xrightarrow{S \rightarrow 0S1} 00S11 \xrightarrow{S \rightarrow 0S1} 000S111 \xrightarrow{S \rightarrow \epsilon} 000111$

Context Free Grammars

Definition

A context-free grammar is a 4-tuple (V, Σ, R, S) , where:

- 1 V is a finite set of variables,
- 2 Σ is a finite set of terminals, $\Sigma \cap V = \emptyset$
- 3 R is a finite set of rules, $\alpha \rightarrow \beta$, where α is a variable and β a string of variables and terminals
- 4 $S \in V$ is the start variable

Example: $G = (\{S\}, \{0, 1\}, R, S)$, where the set of rules R is given by: $S \rightarrow 0S1 \mid \epsilon$

And the language of G is $L = \{0^n 1^n \mid n \geq 0\}$

Context Free Grammars

Definition

If u, v , and w are strings of variables and terminals, and $A \rightarrow w$ is a rule of the grammar, we say that uAv **yields** uwv , written $uAv \Rightarrow uwv$. We say that u **derives** v , written $u \xRightarrow{*} v$, if $u = v$ or if a sequence u_1, u_2, \dots, u_k exists for $k \geq 0$ and $u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_k \Rightarrow v$.

- The **language** of the grammar is $\{w \in \Sigma^* \mid S \xRightarrow{*} w\}$.
- Any language that can be generated by some context-free grammar is called a **context-free language** (CFL).

Example: $S \rightarrow 0S1 \mid \epsilon$

equivalent

$S \rightarrow 0S1$
 $S \rightarrow \epsilon$

$S \rightarrow 0S1$

Context Free Grammars

Definition

If u, v , and w are strings of variables and terminals, and $A \rightarrow w$ is a rule of the grammar, we say that uAv **yields** uwv , written $uAv \Rightarrow uwv$. We say that u **derives** v , written $u \xRightarrow{*} v$, if $u = v$ or if a sequence u_1, u_2, \dots, u_k exists for $k \geq 0$ and $u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_k \Rightarrow v$.

- The **language** of the grammar is $\{w \in \Sigma^* | S \xRightarrow{*} w\}$.
- Any language that can be generated by some context-free grammar is called a **context-free language** (CFL).

Example: $S \rightarrow 0S1 | \epsilon$

Parsing and parse trees: compilers and interpreters use parsers to extract the meaning of a program before generating the compiled code; the parser can be built, given a CFG.

Context-Free Languages and Grammars

$\langle \text{SENTENCE} \rangle \rightarrow \langle \text{NOUN.PHRASE} \rangle \langle \text{VERB.PHRASE} \rangle$
 $\langle \text{NOUN.PHRASE} \rangle \rightarrow \langle \text{CMPLX.NOUN} \rangle | \langle \text{CMPLX.NOUN} \rangle \langle \text{PREP.PHRASE} \rangle$
 $\langle \text{VERB.PHRASE} \rangle \rightarrow \langle \text{CMPLX.VERB} \rangle | \langle \text{CMPLX.VERB} \rangle \langle \text{PREP.PHRASE} \rangle$
 $\langle \text{PREP.PHRASE} \rangle \rightarrow \langle \text{PREP} \rangle \langle \text{CMPLX.NOUN} \rangle$
 $\langle \text{CMPLX.NOUN} \rangle \rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle$
 $\langle \text{CMPLX.VERB} \rangle \rightarrow \langle \text{VERB} \rangle | \langle \text{VERB} \rangle \langle \text{NOUN.PHRASE} \rangle$
 $\langle \text{ARTICLE} \rangle \rightarrow a | the$
 $\langle \text{NOUN} \rangle \rightarrow boy | girl | flower$
 $\langle \text{VERB} \rangle \rightarrow sees | likes$
 $\langle \text{PREP} \rangle \rightarrow with$

A handwritten parse tree for the sentence "a boy sees". The root node is <SENTENCE>, which branches into <NOUN.PHRASE> and <VERB.PHRASE>. <NOUN.PHRASE> branches into <CMPLX.NOUN>, which further branches into <ARTICLE> (labeled "a") and <NOUN> (labeled "boy"). <VERB.PHRASE> branches into <CMPLX.VERB>, which branches into <VERB> (labeled "sees").

Strings generated by this grammar include:

- The boy sees the girl
- The girl likes the boy with the flower
- A boy sees

$\langle \text{SENTENCE} \rangle \Rightarrow \langle \text{NOUN.PHRASE} \rangle \langle \text{VERB.PHRASE} \rangle \Rightarrow$
 $\langle \text{CMPLX.NOUN} \rangle \langle \text{VERB.PHRASE} \rangle \Rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \langle \text{VERB.PHRASE} \rangle \Rightarrow$
 $a \langle \text{NOUN} \rangle \langle \text{VERB.PHRASE} \rangle \Rightarrow a \text{ boy } \langle \text{VERB.PHRASE} \rangle \Rightarrow$
 $a \text{ boy } \langle \text{CMPLX.VERB} \rangle \Rightarrow a \text{ boy } \langle \text{VERB} \rangle \Rightarrow a \text{ boy sees}$

- Context-free grammars are used to specify the syntax of a language.
- Programming languages such as Python and Java have context-free grammars and they are used for parsing.
- The $O(n^3|G|)$ CYK algorithm uses dynamic programming to check whether a given string (program) can be generated by the context-free grammar for the language.

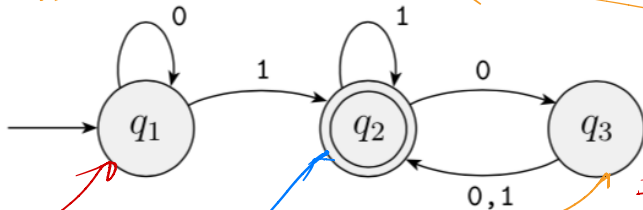
Grammars

- We have seen two different, though equivalent, methods of describing regular languages: FAs and REs
- A grammar provides yet another way to describe a language
- While an automaton *recognizes* a language, a grammar *generates* the language
- The grammars for regular languages are also known as *type 3 grammars*
- The grammars for context-free languages are context-free grammars, also known as *type 2 grammars*
- There exists a finite automaton for a language $L \iff$ there exists a type 3 grammar for L
- There exists a *pushdown automaton* for a language $L \iff$ there exists a type 2 grammar for L

! [NON DETERMINISTIC

Type 3 Grammars: Example

$S \xrightarrow{1} 0S \xrightarrow{2} 01A \xrightarrow{3} 011A \xrightarrow{5} 011$



$S \rightarrow 0S$

$S \rightarrow 1A$

$A \rightarrow 1A$

$A \rightarrow 0B$

$A \rightarrow \epsilon$

$B \rightarrow 0A$

$B \rightarrow 1A$

becomes q2 on next state

$S \rightarrow 0S|1A$

$A \rightarrow 1A|0B|\epsilon$

$B \rightarrow 0A|1A$

$S \xrightarrow{1} 0S \xrightarrow{2} 01A \xrightarrow{3} 011$

01
011
000

$S \xrightarrow{4} 1A \xrightarrow{4} 10B$
 $\xrightarrow{5} 100A \xrightarrow{5} 100$

11S

Designing Grammars

① $L_1 = \{w \mid w \text{ contains at least three 1s}\}$

Designing Grammars

- 1 $L_1 = \{w \mid w \text{ contains at least three 1s}\}$

\approx

$\varepsilon^* | \varepsilon^* 1 \varepsilon^* | \varepsilon^* 1 \varepsilon^* 1 \varepsilon^*$



Designing Grammars

- ① $L_1 = \{w \mid w \text{ contains at least three 1s}\}$

$$S_1 \rightarrow R1R1R1R$$

$$R \rightarrow 0R \mid 1R \mid \epsilon$$

0

000 100
001 101

- ② $L_2 = \{w \mid \text{the length of } w \text{ is odd and its middle symbol is a 0}\}$

00000
xx 0 xx
.
,
,

Designing Grammars

- ① $L_1 = \{w \mid w \text{ contains at least three 1s}\}$

$$S_1 \rightarrow R1R1R1R$$

$$R \rightarrow 0R \mid 1R \mid \epsilon$$

Handwritten examples for L_1 :
 000
 $S_1 \rightarrow 0S_1 \rightarrow 000$
 10010

- ② $L_2 = \{w \mid \text{the length of } w \text{ is odd and its middle symbol is a 0}\}$

$$S_2 \rightarrow 0|0S_2|0S_1|1S_2|1S_1$$

Handwritten annotations: 1, 2, 3, 4, 5 above the symbols; 2, 2, 2, 2 below the symbols.

Handwritten examples for L_2 :
 $S_2 \rightarrow 1S_2 \rightarrow 1S_2 \rightarrow 10S_2 \rightarrow 10S_1 \rightarrow 100$
 $10S_1 \rightarrow 100$

- ③ L_3 is strings over the alphabet $\{a, b\}$ with more a 's than b 's

Designing Grammars

- ① $L_1 = \{w \mid w \text{ contains at least three 1s}\}$

$$S_1 \rightarrow R1R1R1R$$

$$R \rightarrow 0R \mid 1R \mid \epsilon$$

- ② $L_2 = \{w \mid \text{the length of } w \text{ is odd and its middle symbol is a 0}\}$

$$S_2 \rightarrow 0 \mid 0S_2 \mid 0S_1 \mid 1S_2 \mid 1S_1$$

- ③ L_3 is strings over the alphabet $\{a, b\}$ with more a 's than b 's

$$S_3 \xrightarrow{1} TaT$$

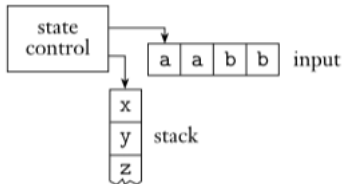
$$T \rightarrow \underset{2}{TT} \mid \underset{3}{aTb} \mid \underset{4}{bTa} \mid \underset{5}{a} \mid \underset{6}{\epsilon}$$

Handwritten derivation for L_3 :

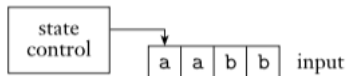
$$\begin{aligned} & baa \\ & S_3 \xrightarrow{1} TaT \xrightarrow{4} bTaT \xrightarrow{6} baT \xrightarrow{5} ba \end{aligned}$$

Pushdown Automata

Schematic Pushdown Automaton

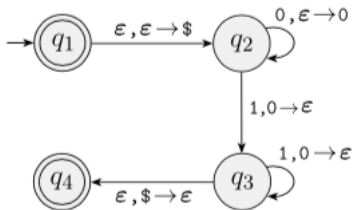
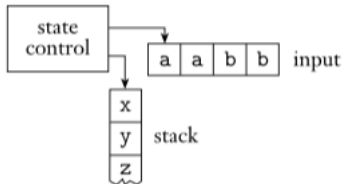


Schematic Finite Automaton

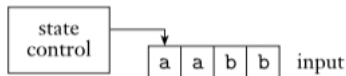


Pushdown Automata

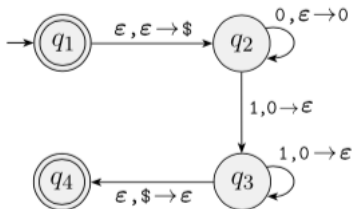
Schematic Pushdown Automaton



Schematic Finite Automaton



$\{0^n 1^n \mid n \geq 0\}$



A pushdown automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_s, F)$, where

- Q is a finite set called the states,
- Σ is a finite input alphabet,
- Γ is a finite stack alphabet,
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow P(Q \times \Gamma_\epsilon)$ is the transition function,
- $q_s \in Q$ is the start state,
- $F \subseteq Q$ is the set of accept states.