Chapter 2
Context-Free Languages

Overview

- so far, we have looked at FA's and regular expressions
  - different, though equivalent
  - some simple languages, such as 0^n1^n cannot be described in these ways

Overview

- we now turn to context-free grammars (CFGs)
  - more powerful way to describe languages
    - can describe recursive structures of languages
  - first used to study human languages
    - relationships between parts of language (noun, verb, etc.) lead to recursion
      - e.g., noun phrases may appear inside verb phrases and vice versa
      - context-free grammars help organize and understand such relationships

Overview

- another important application is in the specification of programming languages
  - a grammar for a programming language can help people learn about the language syntax
  - compiler and interpreter designers often start with the grammar for a programming language
    - parser: extracts meaning from code before execution
    - some tools can automatically generate a parser from the grammar

Overview

- the collection of languages associated with context-free grammars are context-free languages
  - include all regular languages
  - plus other languages

- we will study
  - context-free grammars
  - formal definition of context-free grammars
  - properties of context-free languages
  - pushdown automata: machines recognizing context-free languages
    - help us realize the power of context-free grammars

Context-Free Grammars

- example: CFG G_1

  $A \rightarrow 0A1$

  $A \rightarrow B$

  $B \rightarrow \#$

- a grammar consists of
  - substitution rules, or productions
    - each rule appears as a line in the grammar
    - lhs: variable or nonterminal
    - derivation symbol
    - rhs: variables or terminals
  - common notation
    - nonterminals: capital letters
    - terminals: lowercase letters, numbers, symbols

Context-Free Grammars

- example: CFG G_1

  $A \rightarrow 0A1$

  $A \rightarrow B$

  $B \rightarrow \#$

- start variable: lhs of first production
- G_1 has
  - two variables: A, B
  - start variable: A
  - terminals: 0, 1, #

Context-Free Grammars

- process for generating strings in the language using the grammar
  - write down the start variable
    - lhs of top rule, unless otherwise stated
  - find a variable that is written down and a rule that starts with that variable
    - replace the variable with rhs of that rule
  - repeat replacements until no variable remains

Context-Free Grammars

- example: CFG G_1

  $A \rightarrow 0A1$

  $A \rightarrow B$

  $B \rightarrow \#$

- we can generate string 000#111 with the following derivation (or sequence of substitutions)

  $A \rightarrow 0A1$

  $\rightarrow 00A11$

  $\rightarrow 000A111$

  $\rightarrow 000B111$

  $\rightarrow 000\#111$

Context-Free Grammars

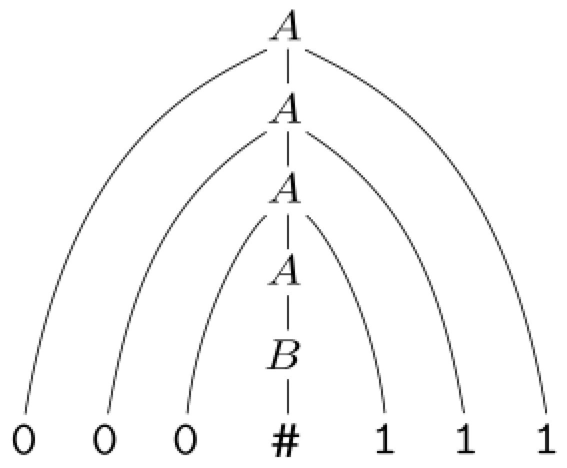
- example: CFG G_1

  $A \rightarrow 0A1$

  $A \rightarrow B$

  $B \rightarrow \#$

- derivation can also be shown with a parse tree



Context-Free Grammars

- all strings generated from derivations constitute the language of the grammar, L(G)
  - $L(G_1) = \{0^n\#1^n \mid n \geq 0\}$
  - any language that can be generated by a CFG is called a context-free language (CFL)

- for convenience, we can replace

$$A \rightarrow 0A1$$
$$A \rightarrow B$$

with

$$A \rightarrow 0A1 \mid B$$

Context-Free Grammars

- example: CFG G_2 describes part of the English language

$$
\begin{aligned}
\langle\text{SENTENCE}\rangle &\rightarrow \langle\text{NOUN-PHRASE}\rangle\langle\text{VERB-PHRASE}\rangle \\
\langle\text{NOUN-PHRASE}\rangle &\rightarrow \langle\text{CMPLX-NOUN}\rangle \mid \langle\text{CMPLX-NOUN}\rangle\langle\text{PREP-PHRASE}\rangle \\
\langle\text{VERB-PHRASE}\rangle &\rightarrow \langle\text{CMPLX-VERB}\rangle \mid \langle\text{CMPLX-VERB}\rangle\langle\text{PREP-PHRASE}\rangle \\
\langle\text{PREP-PHRASE}\rangle &\rightarrow \langle\text{PREP}\rangle\langle\text{CMPLX-NOUN}\rangle \\
\langle\text{CMPLX-NOUN}\rangle &\rightarrow \langle\text{ARTICLE}\rangle\langle\text{NOUN}\rangle \\
\langle\text{CMPLX-VERB}\rangle &\rightarrow \langle\text{VERB}\rangle \mid \langle\text{VERB}\rangle\langle\text{NOUN-PHRASE}\rangle \\
\langle\text{ARTICLE}\rangle &\rightarrow \texttt{a} \mid \texttt{the} \\
\langle\text{NOUN}\rangle &\rightarrow \texttt{boy} \mid \texttt{girl} \mid \texttt{flower} \\
\langle\text{VERB}\rangle &\rightarrow \texttt{touches} \mid \texttt{likes} \mid \texttt{sees} \\
\langle\text{PREP}\rangle &\rightarrow \texttt{with}
\end{aligned}
$$

- 10 variables (nonterminals)
- 27 terminals (26 alphabet letters plus space)
- 18 rules

Context-Free Grammars

$$
\begin{aligned}
\langle\text{SENTENCE}\rangle &\to \langle\text{NOUN-PHRASE}\rangle\langle\text{VERB-PHRASE}\rangle\\
\langle\text{NOUN-PHRASE}\rangle &\to \langle\text{CMPLX-NOUN}\rangle \mid \langle\text{CMPLX-NOUN}\rangle\langle\text{PREP-PHRASE}\rangle\\
\langle\text{VERB-PHRASE}\rangle &\to \langle\text{CMPLX-VERB}\rangle \mid \langle\text{CMPLX-VERB}\rangle\langle\text{PREP-PHRASE}\rangle\\
\langle\text{PREP-PHRASE}\rangle &\to \langle\text{PREP}\rangle\langle\text{CMPLX-NOUN}\rangle\\
\langle\text{CMPLX-NOUN}\rangle &\to \langle\text{ARTICLE}\rangle\langle\text{NOUN}\rangle\\
\langle\text{CMPLX-VERB}\rangle &\to \langle\text{VERB}\rangle \mid \langle\text{VERB}\rangle\langle\text{NOUN-PHRASE}\rangle\\
\langle\text{ARTICLE}\rangle &\to \texttt{a} \mid \texttt{the}\\
\langle\text{NOUN}\rangle &\to \texttt{boy} \mid \texttt{girl} \mid \texttt{flower}\\
\langle\text{VERB}\rangle &\to \texttt{touches} \mid \texttt{likes} \mid \texttt{sees}\\
\langle\text{PREP}\rangle &\to \texttt{with}
\end{aligned}
$$

- strings in $L(G_2)$

  a boy sees

  the boy sees a flower

  a girl with a flower likes the boy

- example derivation of 'a boy sees'

$$
\begin{aligned}
\langle\text{SENTENCE}\rangle &\Rightarrow \langle\text{NOUN-PHRASE}\rangle\langle\text{VERB-PHRASE}\rangle\\
&\Rightarrow \langle\text{CMPLX-NOUN}\rangle\langle\text{VERB-PHRASE}\rangle\\
&\Rightarrow \langle\text{ARTICLE}\rangle\langle\text{NOUN}\rangle\langle\text{VERB-PHRASE}\rangle\\
&\Rightarrow \texttt{a}\ \langle\text{NOUN}\rangle\langle\text{VERB-PHRASE}\rangle\\
&\Rightarrow \texttt{a boy}\ \langle\text{VERB-PHRASE}\rangle\\
&\Rightarrow \texttt{a boy}\ \langle\text{CMPLX-VERB}\rangle\\
&\Rightarrow \texttt{a boy}\ \langle\text{VERB}\rangle\\
&\Rightarrow \texttt{a boy sees}
\end{aligned}
$$

Context-Free Grammars

- formal definition of CFG

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

1. V is a finite set called the ***variables***,

2. Σ is a finite set, disjoint from V, called the ***terminals***,

3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and

4. $S \in V$ is the start variable.

- in grammar G_1
  - V = {A, B}
  - Σ = {0, 1, #}
  - S = A
  - R is the collection of rules in the grammar

Context-Free Grammars

- formal definition of CFG notes
  - for u, v, w (strings of variables and terminals)
  - $A \rightarrow w$ is a rule
  - uAv yields uwv, or $uAv \Rightarrow uwv$
  - u derives v, or $u \Rightarrow^* v$ if $u = v$ or if $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ and $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$

  - the language of the grammar is $\{w \in \Sigma^* \mid S \Rightarrow^* w\}$

Context-Free Grammars

- example: grammar G_2

$$V = \{\langle\text{SENTENCE}\rangle, \langle\text{NOUN-PHRASE}\rangle, \langle\text{VERB-PHRASE}\rangle,$$
$$\langle\text{PREP-PHRASE}\rangle, \langle\text{CMPLX-NOUN}\rangle, \langle\text{CMPLX-VERB}\rangle,$$
$$\langle\text{ARTICLE}\rangle, \langle\text{NOUN}\rangle, \langle\text{VERB}\rangle, \langle\text{PREP}\rangle\},$$

- Σ = {a, b, c, … ,z, " "}
  - " " represents blank space
- can specify grammar by just writing rules
  - identify variables as appearing on lhs
  - all other symbols are terminals
  - start variable is lhs of first rule

Context-Free Grammars

- example: grammar G_3
  - $G_3 = (\{S\}, \{a, b\}, R, S)$
    - rules:

      $S \rightarrow aSb \mid SS \mid \varepsilon$

  - note ε
  - example strings generated:

    abab, aaabbb, aababb, ε
    - can think of a and b as '(' and ')', respectively
      - strings generated are properly nested parentheses

Context-Free Grammars

- example: grammar G_4 = (V, Σ, R, <EXPR>)
  - V = {<EXPR>, <TERM>, <FACTOR>}
  - Σ = {a, +, x, (,)}
  - R (rules) are

    <EXPR> → <EXPR> + <TERM> | <TERM>

    <TERM> → <TERM> x <FACTOR> | <FACTOR>

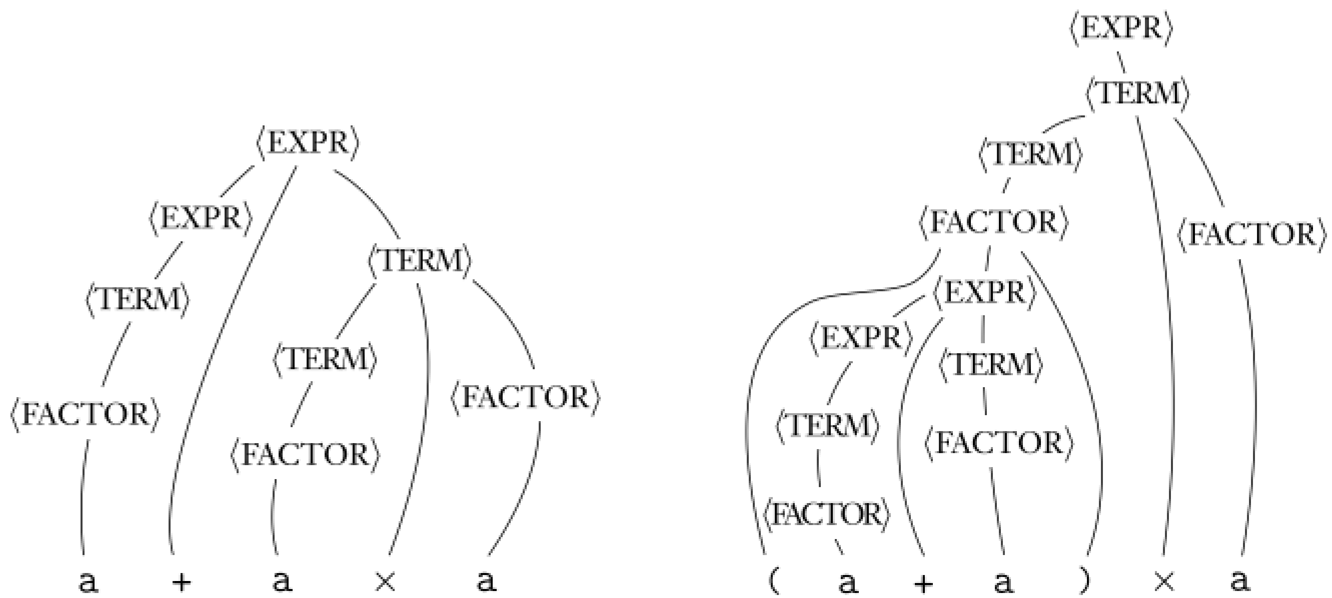
    <FACTOR> → <EXPR> | a

  - describes part of a programming language for arithmetic expressions

Context-Free Grammars

- example: grammar G_4 = (V, Σ, R, <EXPR>)

  - parse trees for strings a+axa and (a+a)xa
    - note precedence imposed



Context-Free Grammars

- designing CFGs
  - requires some creativity, as with FAs

  - helpful techniques
    - first, many CFLs are the union of simpler CFLs
      - construct individual grammars for pieces
      - solving several simpler problems easier than solving one complicated problem
      - merge by combining rules and adding new rule where S_i are the start variables for the simpler grammars

$$S \rightarrow S_1 \mid S_2 \mid \ldots \mid S_k$$

Context-Free Grammars

- designing CFGs (cont.)
  - many CFLs are the union of simpler CFLs

    example: create a grammar for the language

    $\{0^n1^n \mid n \geq 0\} \cup \{1^n0^n \mid n \geq 0\}$

    1. construct grammar for $\{0^n1^n \mid n \geq 0\}$

       $S_1 \rightarrow 0S_11 \mid \varepsilon$

    2. construct grammar for $\{1^n0^n \mid n \geq 0\}$

       $S_2 \rightarrow 1S_20 \mid \varepsilon$

    3. add rule $S \rightarrow S_1 \mid S_2$

       $S \rightarrow S_1 \mid S_2$

       $S_1 \rightarrow 0S_11 \mid \varepsilon$

       $S_2 \rightarrow 1S_20 \mid \varepsilon$

Context-Free Grammars

- designing CFGs (cont.)
  - helpful techniques
    - second, constructing a CFG for a regular language is easy if you can construct a DFA first
      - convert DFA into CFG
        - make a variable R for each state q_i of the DFA
        - add rule $R_i \rightarrow aR_j$ if $\delta(q_i, a) = q_j$ is transition in DFA
        - add rule $R_i \rightarrow \varepsilon$ if q_i is an accept state of the DFA
        - R_0 is the start variable where q_0 is the start state

Context-Free Grammars

- designing CFGs (cont.)
  - helpful techniques
    - third, certain CFLs contain strings with two substrings that are linked in such a way that a FA would need to remember
      - e.g., $\{0^n1^n \mid n \geq 0\}$
      - construct a CFG to handle this situation by using a rule of the form $R \rightarrow uRv$ where the numbers of u's and v's are the same

Context-Free Grammars

- designing CFGs (cont.)
  - helpful techniques
    - finally, in more complex languages, the strings may contain certain structures that appear recursively as part of other (or the same) structures
      - e.g., G_4 that generates arithmetic expressions
        - any time an a appears, an entire parenthesized expression might appear recursively instead
        - place the variable symbol generating the structure in the location of the rules corresponding to where that structure may recursively appear

Context-Free Grammars

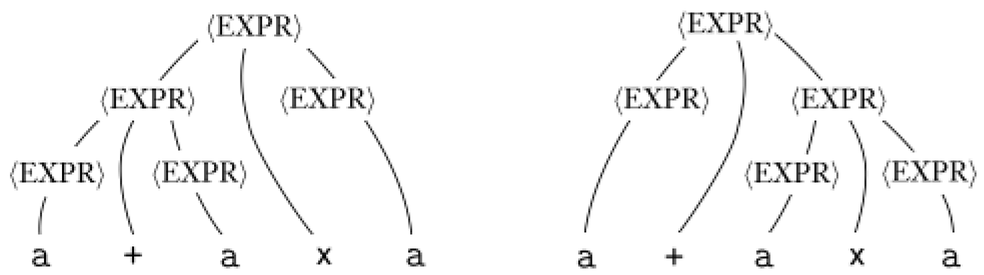
- ambiguity
  - sometimes a grammar can generate a string in several different ways
    - must be different parse trees
    - undesirable in certain applications, such as programming languages, since a program should have only one interpretation
    - string is derived ambiguously
    - if grammar generates strings ambiguously, grammar is ambiguous

Context-Free Grammars

- ambiguity (cont.)
  - example: grammar G_5

    $\langle EXPR \rangle \rightarrow \langle EXPR \rangle + \langle EXPR \rangle \mid \langle EXPR \rangle \times \langle EXPR \rangle \mid (\langle EXPR \rangle) \mid a$

    - generates string a+axa ambiguously



- note precedence

Context-Free Grammars

- ambiguity (cont.)
  - G_4 generates the same language as G_5, but every string has a unique parse tree
    - G_4 is unambiguous whereas G_5 is ambiguous
  - G_2 is ambiguous because the following sentence has two different derivations resulting in different parse trees

    the girl touches the boy with the flower

Context-Free Grammars

- ambiguity (cont.)
  - formally, a grammar is ambiguous if there is more than one parse tree for deriving the same string
    - not just more than one derivation
      - derivations may differ only in order of replacements, not structure
      - we can focus on structure by replacing variables in a fixed order
        - leftmost derivation: replace the leftmost variable in each step of the derivation

A string w is derived ***ambiguously*** in context-free grammar G if it has two or more different leftmost derivations. Grammar G is ***ambiguous*** if it generates some string ambiguously.

Context-Free Grammars

- ambiguity (cont.)
  - sometimes we can find an unambiguous grammar that generates the same language as an ambiguous one
  - some CFLs can only be generated by ambiguous grammars

Context-Free Grammars

- Chomsky normal form
  - convenient to have CFGs in simplified form

A context-free grammar is in ***Chomsky normal form*** if every rule is of the form

$$A \to BC$$
$$A \to a$$

where a is any terminal and A, B, and C are any variables—except that B and C may not be the start variable. In addition, we permit the rule $S \to \varepsilon$, where S is the start variable.

Context-Free Grammars

- Chomsky normal form
  - any context-free language is generated by a context-free grammar in Chomsky normal form
    - proof idea
      - conversion has several stages where rules that violate the conditions are replaced with equivalent ones that fulfill the requirements
        - add a new start variable
        - eliminate all ε-rules of the form $A \rightarrow \varepsilon$
        - eliminate all unit rules $A \rightarrow B$
        - convert remaining rules into proper form
        - verify that new grammar generates same language

Context-Free Grammars

- Chomsky normal form (cont.)
  - any context-free language is generated by a context-free grammar in Chomsky normal form
    - proof
      - add a new start variable, S_0 and the rule $S_0 \rightarrow S$ where S was the original start state
        - guarantees that the start variable does not appear on the rhs of a rule

Context-Free Grammars

- Chomsky normal form (cont.)
  - any context-free language is generated by a context-free grammar in Chomsky normal form
    - proof
      - eliminate all ε-rules of the form $A \to \varepsilon$ (where A is not the start variable)
        - wherever A appears on the rhs of a rule, add a new rule with that occurrence deleted
        - if $R \to uAv$ is a rule, add rule $R \to uv$
        - add rule for each occurrence of A, so $R \to uAvAw$ results in adding $R \to uvAw$, $R \to uAvw$, and $R \to uvw$
        - if we had $R \to A$, add $R \to \varepsilon$ unless we already removed that rule
        - repeat until all ε-rules removed not using start var

Context-Free Grammars

- Chomsky normal form (cont.)
  - any context-free language is generated by a context-free grammar in Chomsky normal form
    - proof
      - remove unit rules $A \rightarrow B$
        - when a rule $B \rightarrow u$ appears, add rule $A \rightarrow u$ unless this unit rule was previously removed
        - repeat until all unit rules removed

Context-Free Grammars

- Chomsky normal form (cont.)
  - any context-free language is generated by a context-free grammar in Chomsky normal form
    - proof
      - convert all remaining rules into proper form
        - replace rule $A \to u_1u_2 \ldots u_k$ where $k \geq 3$ and each u_i is a variable or terminal symbol with the rules $A \to u_1A_1$, $A_1 \to u_2A_2$, $A_2 \to u_3A_3$, … $A_{k-2} \to u_{k-1}u_k$
        - A_i's are new variables
        - replace any terminal u_i in the preceding rule(s) with new variable U_i and add rule $U_i \to u_i$

Context-Free Grammars

- example: convert the CFG G_6 into Chomsky normal form
  1. add new start state (old on left; new on right)

$$
\begin{aligned}
S &\to ASA \mid aB \\
A &\to B \mid S \\
B &\to b \mid \varepsilon
\end{aligned}
\qquad
\begin{aligned}
\boldsymbol{S_0} &\to \boldsymbol{S} \\
S &\to ASA \mid aB \\
A &\to B \mid S \\
B &\to b \mid \varepsilon
\end{aligned}
$$

  2. remove ε-rules: B → ε on left, A → ε on right

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid aB \mid \mathbf{a} \\
A &\to B \mid S \mid \boldsymbol{\varepsilon} \\
B &\to b \mid \varepsilon
\end{aligned}
\qquad
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid aB \mid a \mid \boldsymbol{SA} \mid \boldsymbol{AS} \mid \boldsymbol{S} \\
A &\to B \mid S \mid \varepsilon \\
B &\to b
\end{aligned}
$$

Context-Free Grammars

- example: convert the CFG G_6 into Chomsky normal form

3a. remove unit rules: $S \to S$ on left, and $S_0 \to S$ on right

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid aB \mid a \mid SA \mid AS \mid S \\
A &\to B \mid S \\
B &\to b
\end{aligned}
$$

$$
\begin{aligned}
S_0 &\to S \mid \mathbf{ASA} \mid \mathbf{aB} \mid \mathbf{a} \mid \mathbf{SA} \mid \mathbf{AS} \\
S &\to ASA \mid aB \mid a \mid SA \mid AS \\
A &\to B \mid S \\
B &\to b
\end{aligned}
$$

3b. remove unit rules: $A \to B$ on left, and $A \to S$ on right

$$
\begin{aligned}
S_0 &\to ASA \mid aB \mid a \mid SA \mid AS \\
S &\to ASA \mid aB \mid a \mid SA \mid AS \\
A &\to B \mid S \mid \mathbf{b} \\
B &\to b
\end{aligned}
$$

$$
\begin{aligned}
S_0 &\to ASA \mid aB \mid a \mid SA \mid AS \\
S &\to ASA \mid aB \mid a \mid SA \mid AS \\
A &\to S \mid b \mid \mathbf{ASA} \mid \mathbf{aB} \mid \mathbf{a} \mid \mathbf{SA} \mid \mathbf{AS} \\
B &\to b
\end{aligned}
$$

Context-Free Grammars

- example: convert the CFG G_6 into Chomsky normal form
  4. convert remaining rules into proper form by adding variables and rules; final grammar is equivalent to G_6; final grammar simplified

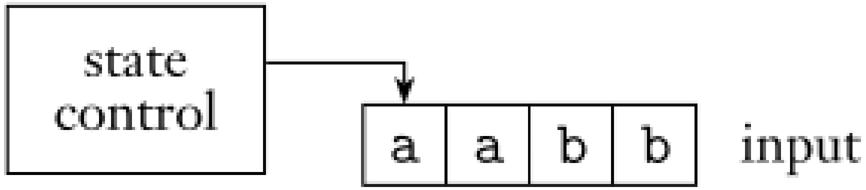
$$
\begin{aligned}
S_0 &\rightarrow AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\
S &\rightarrow AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\
A &\rightarrow \mathtt{b} \mid AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\
A_1 &\rightarrow SA \\
U &\rightarrow \mathtt{a} \\
B &\rightarrow \mathtt{b}
\end{aligned}
$$

Pushdown Automata

- pushdown automata (PDA)
  - like NFA, but includes a stack
    - provides additional memory
    - therefore allows PDA to recognize some nonregular languages
  - equivalent to CFGs
    - now we have two options for proving a language is context-free by providing either
      - CFG generating the language
      - PDA recognizing the language
    - some languages are more easily described by generators, while others by recognizers

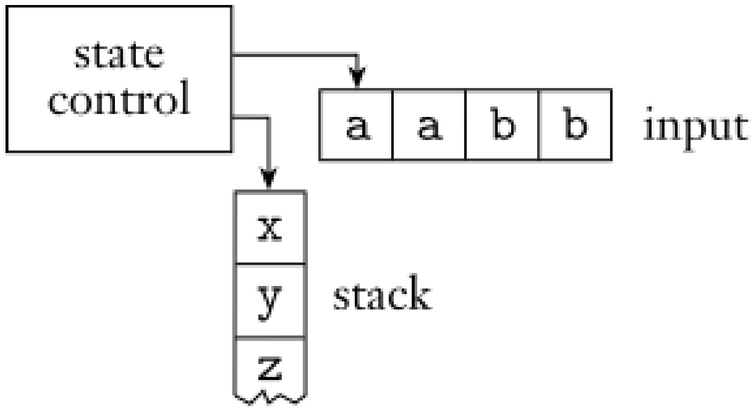
Pushdown Automata

- schematic of a finite automaton
  - control represents states and transition function
  - tape contains the input string
  - arrow is the input head, which points at the next input symbol to be read



Pushdown Automata

- schematic of a PDA
  - add a stack to preceding schematic



Pushdown Automata

- PDA can write symbols on the stack and read them back later
  - writing a symbol pushes other symbols on the stack
  - reading a symbol pops it from the stack
  - all access to the stack takes place at the top (LIFO)
    - analogy: cafeteria plate dispenser

Pushdown Automata

- stacks are useful in PDAs
  - hold an unlimited amount of information
  - FAs typically have very little memory
  - example: the language $\{0^n1^n \mid n \geq 0\}$ cannot be recognized by a FA, but can by a PDA
    - PDA uses its stack to store the number of 0s it has seen (can store numbers of unlimited size)
      - push 0s on the stack as they are read
      - pop off a 0 for each 1 that is read
      - if reading is finished exactly when the stack becomes empty, accept
      - if empty while 1s remain, or if 1s are finished and 0s remain on stack, or 0s in input after the 1s, reject

Pushdown Automata

- nondeterministic PDAs
  - not equivalent in power to deterministic PDAs
  - recognize certain languages no deterministic PDA can
  - DFAs and NFAs recognize the same class of languages
    - so, PDAs are different
  - our focus is on nondeterministic PDAs since they are equivalent in power to CFGs

Pushdown Automata

- PDA formal definition
  - similar to FA, except for the stack
  - stack: device containing symbols from an alphabet
    - may be different from symbols in input
    - input alphabet: Σ
    - stack alphabet: Γ

Pushdown Automata

- PDA formal definition (cont.)
  - transition function
    - $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$
    - $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$
    - domain: $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$
    - current state, next input symbol read, and top symbol of the stack determine the next transition
    - either symbol may be ε, causing the machine to move without reading a symbol from the input or from the stack

Pushdown Automata

- PDA formal definition (cont.)
  - transition function
    - what can the automaton do in transitions?
      - enter a new state and write a symbol on the stack
      - δ can return a member of Q and a member of Γ_ε
    - domain: $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$
    - due to nondeterminism, several legal next moves may be possible
      - a set of members from $Q \times \Gamma_\varepsilon$ may be returned
      - i.e., a member of $P(Q \times \Gamma_\varepsilon)$
    - therefore, $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$

Pushdown Automata

- PDA formal definition (cont.)

A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta \colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

Pushdown Automata

- PDA formal definition (cont.)
  - a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows
    - accepts input w if it can be written as

      $w = w_1 w_2 \ldots w_m$ where each $w_i \in \Sigma_\varepsilon$

      and sequences of states

      $r_0, r_1, \ldots, r_m \in Q$

      and strings

      $s_0, s_1, \ldots, s_m \in \Gamma$ (sequence of stack contents)

      exist that satisfy the following three conditions
      - $r_0 = q_0$ and $s_0 = \varepsilon$
        - i.e., M starts at the start state with an empty stack
      - for $i = 0, \ldots, m-1$, $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ where $s_i = at$ and $s_{i+1} = bt$ for some $a,b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$
        - i.e., M moves properly according to state, stack, and next input symbol
      - $r_m \in F$
        - i.e., an accept state occurs at the end of input

Pushdown Automata

- example: PDA that recognizes $\{0^n1^n \mid n \geq 0\}$
  - let $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, F)$ where
    - $Q = \{q_1, q_2, q_3, q_4\}$
    - $\Sigma = \{0, 1\}$
    - $\Gamma = \{0, \$\}$
    - $F = \{q_1, q_4\}$
    - δ is given by the table where blank entries are Ø

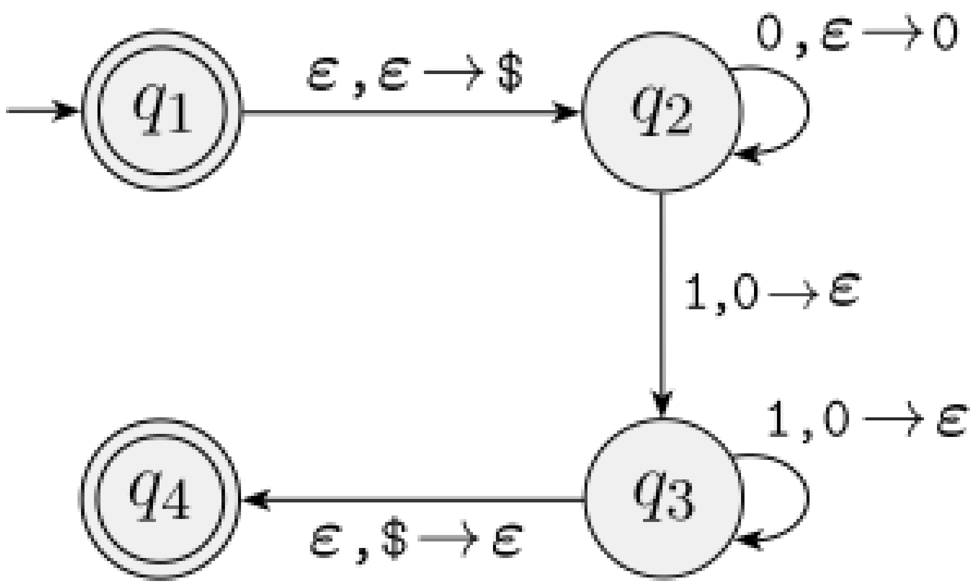
Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

Pushdown Automata

- example: PDA that recognizes $\{0^n1^n \mid n \geq 0\}$
  - we can use a state diagram to describe the PDA
    - similar to state diagrams for FA, but modified for stack updates
      - $a,b \rightarrow c$ means when a is read from input, it may replace b on the top of the stack with c
      - a,b, or c may be ε
        - if $a = \varepsilon$, no symbol read from input
        - if $b = \varepsilon$, no symbol popped from stack
        - if $c = \varepsilon$, no symbol written on stack

Pushdown Automata

- example: PDA that recognizes $\{0^n1^n \mid n \geq 0\}$ (cont.)
  - state diagram



Pushdown Automata

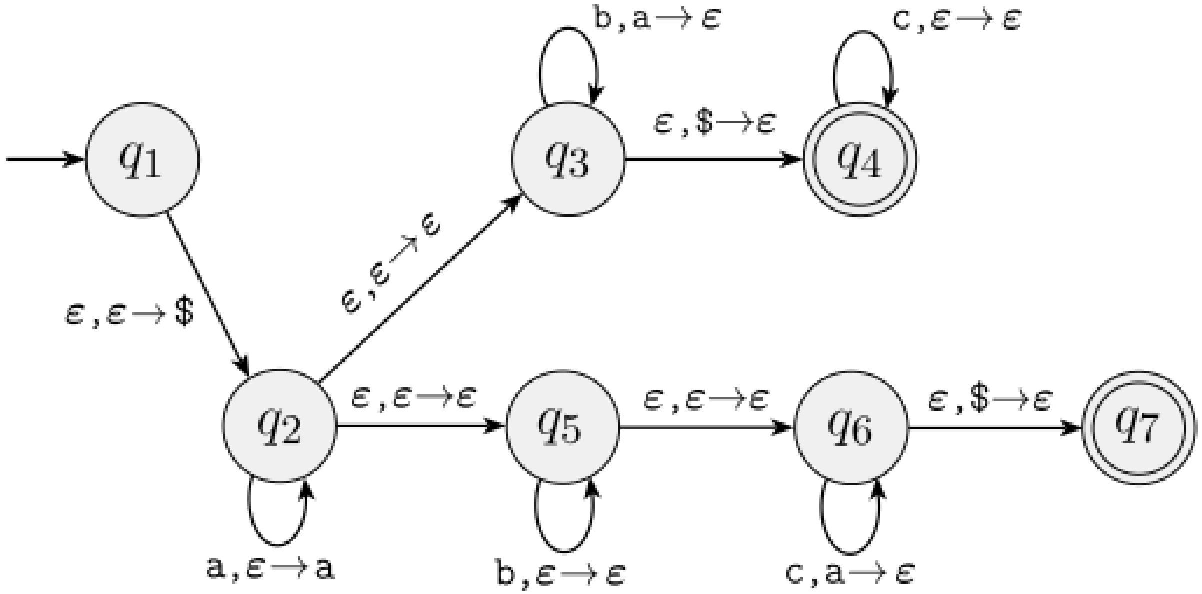
- PDA formal definition contains no test for empty stack
  - instead, initially place a $ on the stack
  - if $ is seen again, the stack is empty
- PDAs cannot test explicitly for reaching end of input string
  - accept state takes effect only when machine is at end of input
  - thus, we assume that PDAs can check for end of input

Pushdown Automata

- example: PDA that recognizes
  $\{a^i b^j c^k \mid i,j,k \geq 0 \text{ and } i=j \text{ or } i=k\}$
  - first read and push a's
  - now it can match them with the b's or c's
    - but don't know which to match
    - using nondeterminism, PDA can guess whether to match b's or c's
      - use two branches: one for each possible guess
      - if either matches, that branch accepts

Pushdown Automata

- example: PDA that recognizes
  $\{a^i b^j c^k \mid i,j,k \geq 0 \text{ and } i=j \text{ or } i=k\}$ (cont.)
  - state diagram

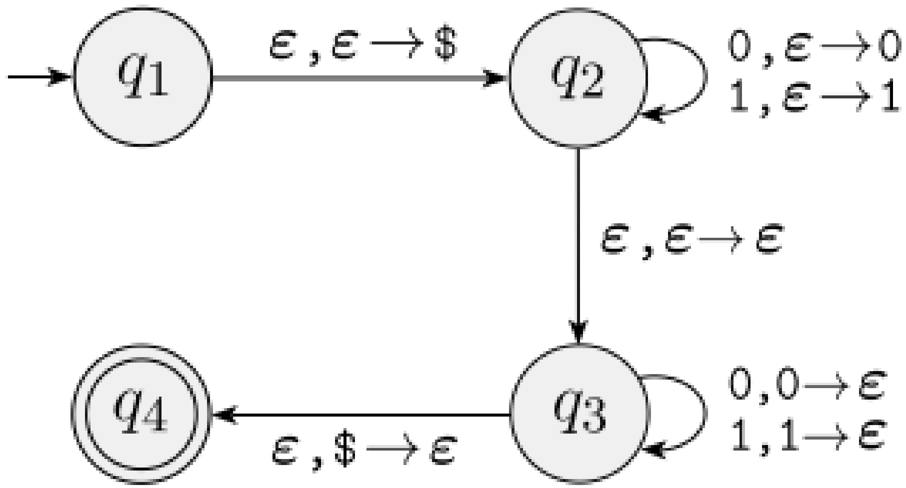


Pushdown Automata

- example: PDA M_3 recognizes
  $\{ww^R \mid w \in \{0, 1\}^*\}$
  - w^R means w written backwards
  - begin by pushing read symbols on stack
  - at each point, nondeterministically guess that the middle of the string has been reached
    - change into popping off the stack for each symbol
    - check to see if popped symbol is the same as read symbol
    - if all are the same, and stack empties when input is finished, accept
    - otherwise, reject

Pushdown Automata

- example: PDA M_3 recognizes
  $\{ww^R \mid w \in \{0, 1\}^*\}$
  - state diagram



Pushdown Automata

- context-free grammars and pushdown automata are equivalent in power
  - both capable of describing class of context-free languages
  - can convert any CFG into a PDA and vice versa
  - recall that a CFL is any language that can be described with a CFG

Pushdown Automata

- theorem: a language is context-free if and only if some PDA recognizes it
  - for if and only if, we have to prove in both directions

Pushdown Automata

- lemma: if a language is context-free, a PDA recognizes it
  - proof idea
    - let A be a CFL
    - therefore, a CFG G generates it
    - convert G into equivalent PDA P
      - P will accept input w if G generates it by determining if there is a derivation for w
        - derivation: a sequence of substitutions made as a grammar generates a string
        - each step yields an intermediate string of variables and terminals
        - P determines whether some series of substitutions from G can lead from the start variable to w

Pushdown Automata

- lemma: if a language is context-free, a PDA recognizes it
  - proof idea (cont.)
    - a difficulty in testing if a derivation for w exists is figuring out which substitutions to make
    - PDA's nondeterminism allows it to guess the sequence of correct substitutions
    - for each step, one of the rules for a particular variable is selected nondeterministically for the substitution

Pushdown Automata

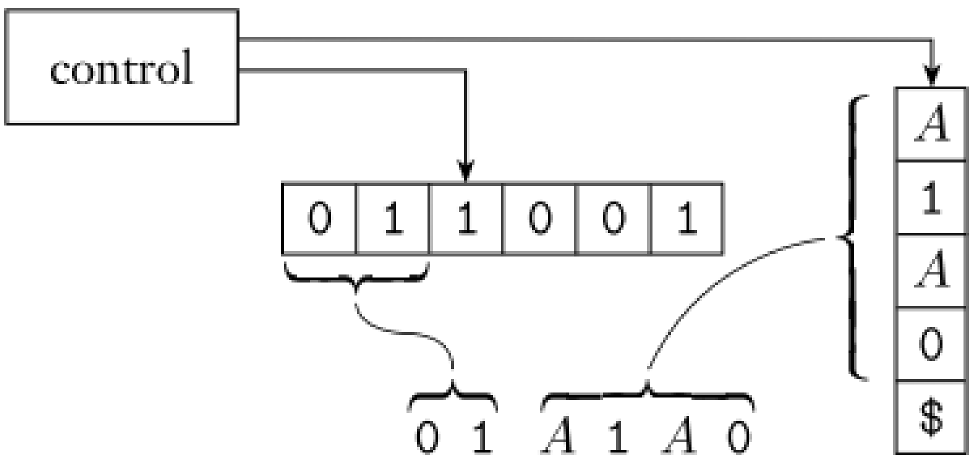
- lemma: if a language is context-free, a PDA recognizes it
  - proof idea (cont.)
    - P begins by writing the start variable on its stack
    - P then goes through intermediate strings, making substitutions
    - if it arrives at a string with only terminal symbols, it has derived a string in the language
    - P accepts this string if it is identical to the one it received as input

Pushdown Automata

- lemma: if a language is context-free, a PDA recognizes it
  - proof idea (cont.)
    - how does the PDA store the intermediate strings as it goes from one state to another?
      - could just store it on the stack
      - won't work because P needs to find variables to replace and make substitutions
      - PDA can only access the top symbol on the stack, which may just be a terminal
    - instead, keep only part of the string on the stack
      - the symbols starting with the first variable in the intermediate string
      - any terminals before the first variable are matched immediately with symbols in the input string

Pushdown Automata

- lemma: if a language is context-free, a PDA recognizes it
  - proof idea (cont.)
    - P representing the intermediate string 01A1A0



Pushdown Automata

- lemma: if a language is context-free, a PDA recognizes it
  - proof idea (cont.)
    - informal description of processing in P
      - push marker symbol $ and start variable on stack
      - repeat the following forever
        - if the top of the stack is a variable symbol *A*, nondeterministically select one of the rules for *A* and substitute with the rhs of the rule
        - if the top of the stack is a terminal a, read the next symbol from the input and compare it to a; if they match, repeat; otherwise, reject on this branch of nondeterminism
        - if the top of the stack is $, enter accept state

Pushdown Automata

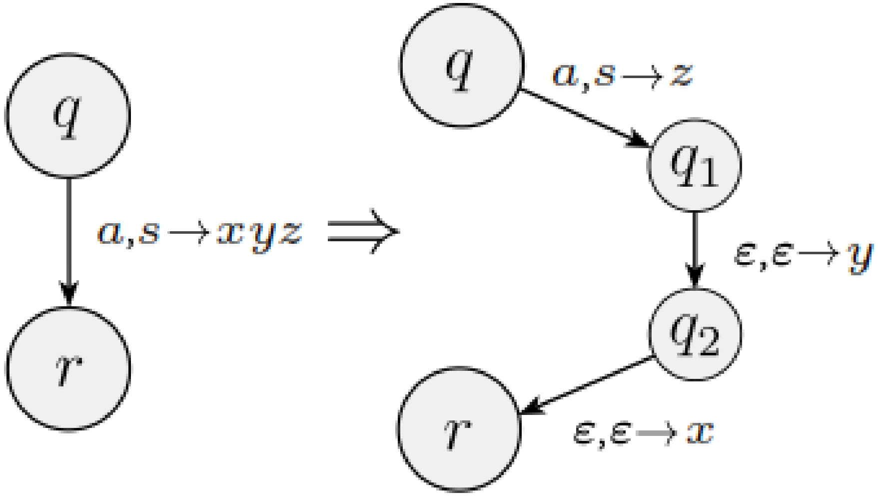
- lemma: if a language is context-free, a PDA recognizes it
  - proof
    - let $P = (Q, \Sigma, \Gamma, \delta, q_{start}, F)$
    - for clarity, use shorthand notation for δ
      - provides a way to write an entire string on the stack in one step
      - simulate by adding states to write the string one symbol at a time
        - let q and r be states of the PDA
        - let $a \in \Sigma_\varepsilon$ and $s \in \Gamma_\varepsilon$
        - go from q to r when a is read and s is popped
        - push string $u = u_1 \ldots u_l$ on stack at the same time

Pushdown Automata

- lemma: if a language is context-free, a PDA recognizes it
  - proof (cont.)
    - implement by adding new states $q_1,\ldots q_{l-1}$ and setting the transition function as follows
      - $\delta(q, a, s)$ to contain (q_1, u_l)
      - $\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, u_{l-1})\}$
      - $\delta(q_2, \varepsilon, \varepsilon) = \{(q_3, u_{l-2})\}$ …
      - $\delta(q_{l-1}, \varepsilon, \varepsilon) = \{(r, u_1)\}$
    - $(r, u) \in \delta(q, a, s)$ means when q is the state of the automaton, a is the next input symbol and s is the symbol on top of the stack
    - PDA may read a and pop s, then push u on the stack and go to state r

Pushdown Automata

- lemma: if a language is context-free, a PDA recognizes it
  - proof (cont.)
    - implementing shorthand $(r, xyz) \in \delta(q, a, s)$



Pushdown Automata

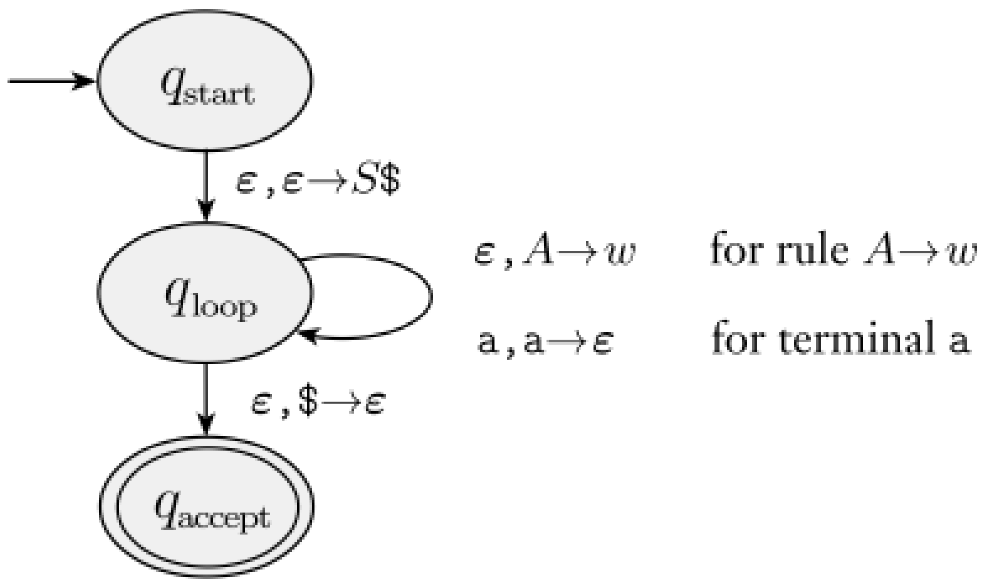
- lemma: if a language is context-free, a PDA recognizes it
  - proof (cont.)
    - $P = (Q, \Sigma, \Gamma, \delta, q_{start}, F)$ where
      - $Q = \{q_{start}, q_{loop}, q_{accept}\} \cup E$
        - E is the states needed for implementing shorthand
      - q_{start} is the start state
      - $F = \{q_{accept}\}$

Pushdown Automata

- lemma: if a language is context-free, a PDA recognizes it
  - proof (cont.)
    - $P = (Q, \Sigma, \Gamma, \delta, q_{start}, F)$ where
      - δ is defined as follows
        - $\delta(q_{start}, \varepsilon, \varepsilon) = \{(q_{loop}, S\$)\}$
          - initialize stack to contain $ and S, implementing step 1 in the informal description
        - $\delta(q_{loop}, \varepsilon, A) = \{(q_{loop}, w) \mid \text{where } A \to w \text{ is a rule in } R\}$
          - the top of the stack contains a variable
        - $\delta(q_{loop}, a, a) = \{(q_{loop}, \varepsilon)\}$
          - the top of the stack contains a terminal
        - $\delta(q_{loop}, \varepsilon, \$) = \{(q_{accept}, \varepsilon)\}$
          - empty stack marker $ is on the top of the stack

Pushdown Automata

- lemma: if a language is context-free, a PDA recognizes it
  - proof (cont.)
    - state diagram of P

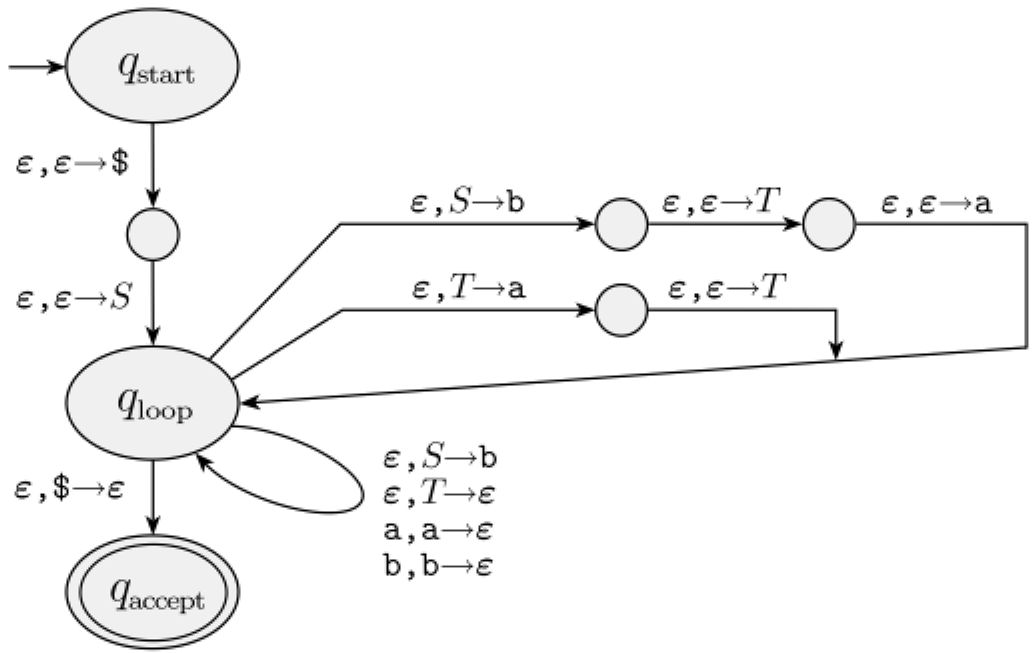


Pushdown Automata

- lemma: if a language is context-free, a PDA recognizes it
  - example: use the procedure to construct a PDA P from the following CFG G

$$S \rightarrow aTb \mid b$$

$$T \rightarrow Ta \mid \varepsilon$$

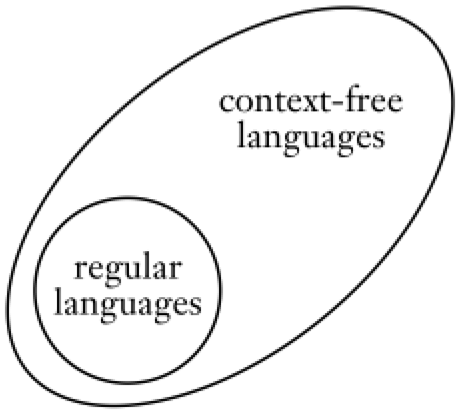


Pushdown Automata

- lemma: if a PDA recognizes a language, it is context-free
  - proof idea
    - harder
    - we have PDA P and want to make CFG G that generates all the strings P accepts
      - or, G should generate a string if it causes the PDA to go from its start state to an accept state

Pushdown Automata

- we have shown that PDAs recognize the class of CFLs
  - we can now establish a relationship between the regular languages and the CFLs
  - since every regular language is recognized by a FA and every FA is automatically a PDA that ignores its stack, every regular language must also be a CFL



Non-Context-Free Languages

- we want to able to prove that some languages are non-context-free
  - use pumping lemma for CFLs
    - every CFL has a pumping length such that all longer strings in the language can be pumped
      - string divided into 5 parts
      - 2nd and 4th parts may be repeated together any number of times with the resulting string in the language

Non-Context-Free Languages

- pumping lemma for CFLs
  - if A is a context-free language, there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into five pieces, s = uvxyz, satisfying the following conditions
    - for each $i \geq 0$, $uv^ixy^iz \in A$
    - $|vy| > 0$
    - $|vxy| \leq p$

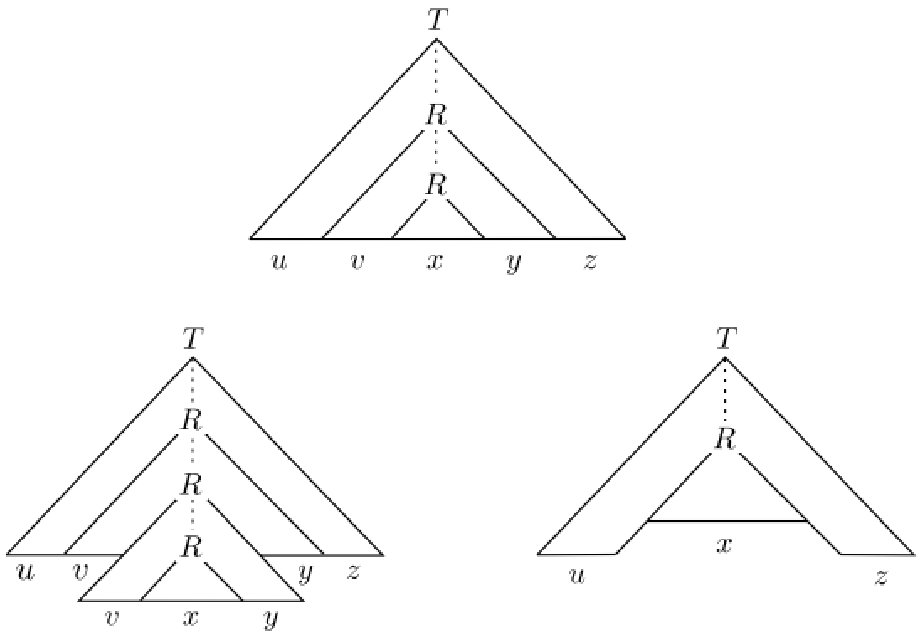
  - condition 2: either v or y is not ε
    - otherwise, theorem trivially true
  - condition 3: max length useful in proving certain languages are not context-free

Non-Context-Free Languages

- pumping lemma for CFLs
  - proof idea
    - let A be a CFL and G be a CFG that generates it
    - show that any sufficiently long string s in A can be pumped and remain in A
    - let s be a very long string in A
    - s is derivable from G and therefore has a parse tree
      - parse tree is very tall because s is very long
      - parse tree contains some long path from the start variable at the root of the tree to one of the terminal symbols at a leaf
      - on this path, some R must repeat due to the pigeonhole principle

Non-Context-Free Languages

- pumping lemma for CFLs
  - proof idea
    - this repetition allows us to replace the subtree under the second R with the subtree under the first R
    - therefore, we can cut s into 5 pieces uvxyz and repeat the 2nd and 4th pieces to obtain a string in the language



Non-Context-Free Languages

- pumping lemma for CFLs
  - example: use the pumping lemma to show $B = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free
    - assume B is context-free with pumping length p
    - select string $a^p b^p c^p$
      - s is a member of B and of length at least p
      - show that no matter how we divide s into uvxyz, one of the three conditions of the lemma is violated
      - condition 2 ensures that either v or y is not ε – consider 2 cases
        - when both v and y contain only one type of symbol, v does not contain both a's and b's or b's and c's, and same for y
          - the string uv^2xy^2z cannot contain an equal number of a's, b's, and c's
        - when either v or y contain more than one type of symbol
          - the string uv^2xy^2z can contain an equal number of a's, b's, and c's, but in the wrong order
      - one of these cases must occur, but both result in a contradiction
      - therefore, B is not a CFL

Non-Context-Free Languages

- pumping lemma for CFLs
  - example: use the pumping lemma to show $B = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free
    - example strings to help explain proof
    - select string $a^p b^p c^p$ if p = 3, string is aaabbbccc
      - since vxy must be <= 3, v and y are
        - both contain one type of symbol: both a's, both b's, or both c's
        - both contain one type of symbol: v is a's and y is b's, or v is b's and y is c's
        - v or y contain more than one type of symbol: v or y straddles a boundary so v is a's and y is b's and c's, v is a's and b's and y is b's, v is b's and y is b's and c's, or v is b's and c's and y is c's
        - either v or y is ε and the other is not (won't work because all three need to increase in number, but this will allow only two, at most, to do so)
    - condition 2 ensures that either v or y is not ε – consider 2 cases
      - when both v and y contain only one type of symbol, v does not contain both a's and b's or b's and c's, and same for y
        - the string uv^2xy^2z cannot contain an equal number of a's, b's, and c's
          - e.g., v=a, y=b: aaaabbbbccc
      - when either v or y contain more than one type of symbol
        - the string uv^2xy^2z can contain an equal number of a's, b's, and c's, but in the wrong order (e.g., v=ab, y=b: aaababbbbccc)

Non-Context-Free Languages

- pumping lemma for CFLs
  - example: show $C = \{a^i b^j c^k \mid 0 \le i \le j \le k\}$ is not context-free
    - assume C is context-free with pumping length p
    - select string $a^p b^p c^p$, but must pump down as well as pump up
      - s is a member of C and of length at least p
      - show that no matter how we divide s into uvxyz, one of the three conditions of the lemma is violated
      - condition 2 ensures that either v or y is not ε – consider 2 cases
        - when both v and y contain only one type of symbol, v does not contain both a's and b's or b's and c's, and same for y
          - one of the symbols does not appear in v or y
          - three subcases
            - a's do not appear: try pumping down to $uv^0xy^0z = uxz$
              - contains same number of a's as s, but fewer b's or fewer c's
            - b's do not appear: try pumping down to $uv^0xy^0z = uxz$
              - either a's or c's must appear in v or y because both can't be ε
              - if a's appear, uv^2xy^2z has more a's than b's
              - if c's appear, uv^0xy^0z has more b's than c's
            - c's do not appear
              - uv^2xy^2z contains more a's or more b's than c's
        - when either v or y contain more than one type of symbol
          - uv^2xy^2z will not contain symbols in the correct order
      - one of these cases must occur, but all result in a contradiction
      - therefore, C is not a CFL

Non-Context-Free Languages

- pumping lemma for CFLs
  - example: use the pumping lemma to show D = {ww| w ∈ {0,1}*} is not context-free
    - assume D is context-free with pumping length p
    - select string 0^p10^p1
      - s is a member of D and of length at least p
      - but this string can be pumped

$$\overbrace{\underbrace{000\cdots000}_{u}\;\underbrace{0}_{v}\;\underbrace{1}_{x}}^{0^p1}\;\overbrace{\underbrace{0}_{y}\;\underbrace{000\cdots0001}_{z}}^{0^p1}$$

Non-Context-Free Languages

- pumping lemma for CFLs
  - example: use the pumping lemma to show $D = \{ww \mid w \in \{0,1\}^*\}$ is not context-free
    - assume D is context-free with pumping length p
    - select string $0^p1^p0^p1^p$
      - s is a member of D and of length at least p
      - by condition 3, $|vxy| \leq p$
      - show that no matter how we divide s into uvxyz, one of the three conditions of the lemma is violated
        - vxy must straddle the midpoint of s; otherwise, pumping s in the first half of the string up to uv^2xy^2z moves a 1 into the first position of the second half
        - if vxy occurs in the second half of s, uv^2xy^2z moves a 0 into the last position of the first half, so no longer in form ww
        - if vxy straddles the midpoint, pumping down to uv^0xy^0z results in $0^p1^i0^j1^p$ where i and j can't both be p, hence not ww
      - all cases result in contradiction
      - therefore, D is not a CFL

Non-Context-Free Languages

- properties of CFLs
  - the class of CFLs is closed under
    - union
      - if A and B are context-free, so is $A \cup B$
    - concatenation
      - if A and B are context-free, so is AB
    - star
      - if A is context-free, so is A^*
    - reverse
      - if A is context-free, so is A^R

Non-Context-Free Languages

- properties of CFLs
  - the class of CFLs is not closed under
    - intersection
      - consider $A = \{a^n b^n c^m\}$ and $B = \{a^m b^n c^n\}$
    - complementation
      - note that $A \cap B = \overline{\overline{A} \cup \overline{B}}$
    - difference
      - note that $\overline{A} = \Sigma^* - A$

Non-Context-Free Languages

- properties of CFLs
  - intersection with a regular language
    - if A is context-free and B is regular, then $A \cap B$ is context-free
  - difference from a regular language
    - if A is context-free and B is regular, then $A - B$ is context-free
    - note that $A - B = A \cap \overline{B}$

Non-Context-Free Languages

- properties of CFLs
  - use the previous properties to prove a language is context-free
  - use the previous properties to prove a language is non-context-free if it does not pass the closure rules
    - example: prove A is not a CFL where

      $A = \{w \in \{a,b,c\}^* \mid w$ has an equal number of a's, b's, and c's$\}$

    consider $L = a^*b^*c^*$ (regular)

    if A is a CFL, $A \cap L = a^ib^ic^i$ should be a CFL, but it's not

Deterministic Context-Free Languages

- recall that DFAs and NFAs are equivalent in power
- but nondeterministic PDAs are more powerful than deterministic PDAs
  - certain CFLs cannot be recognized by DPDAs
  - languages that can be recognized by DPDAs are called deterministic context-free languages (DCFLs)
    - useful in parsers for programming languages