Theory of Computing 2024: Context-Free Languages

(Based on [Sipser 2006, 2013])

Yih-Kuen Tsay

March 18, 2024

1 Context-Free Grammars

Introduction

- We have seen languages that cannot be described by any regular expression (or recognized by any finite automaton).
- *Context-free grammars* are a more powerful method for describing languages; they were first used in the study of natural languages.
- They play an important role in the specification and compilation of programming languages.
- The collection of languages associated with context-free grammars are called the *context-free languages* (CFLs).

Context-Free Grammars

- A *context-free grammar* (CFG) consists of a collection of *substitution rules* (or *productions*) such as:

$$\begin{array}{rcl} A & \rightarrow & 0A1 \\ A & \rightarrow & B \\ B & \rightarrow & \# \end{array} \quad \text{or alternatively} \quad \begin{array}{rcl} A & \rightarrow & 0A1 \mid B \\ B & \rightarrow & \# \end{array}$$

- Symbols A and B here are called *variables*; the other symbols 0, 1, and # are called *terminals*.
- A grammar describes a language by *generating* each string of the language through a *derivation*.

  For example, the above grammar generates the string 000#111:

  $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$.

/* Note that in a derivation the variable A may become, according to the production $A \rightarrow 0A1 \mid B$ (which has a single A on the left-hand side), either $0A1$ or B, the choice of which is independent of whatever comes before or after A, i.e., independent of the context of A. This is why such grammars are called context-free grammars.

There are also *context-sensitive* grammars, where the left-hand side and the right-hand side of a production may be surrounded by a context of variables and terminals, in the form of $\alpha A\beta \rightarrow \alpha\gamma\beta$ (where γ must be non-empty except when A is the start variable/symbol). Context-sensitive grammars (with the additional power of enforcing contexts in a derivation) are more expressive than context-free grammars.

The most general kind of grammars is an *unrestricted* grammar, which only requires the left-hand of a production rule to be non-empty. */

Context-Free Grammars (cont.)

The preceding derivation of 000#111 may be represented pictorially as a *parse tree*:

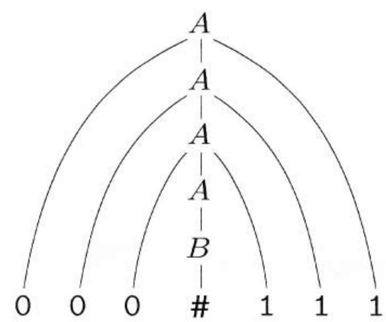

FIGURE 2.1
Parse tree for 000#111 in grammar G_1

Source: [Sipser 2006]

An Example CFG

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

An Example CFG (cont.)

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

Definition of a CFG

Definition 1 (2.2)**.** A **context-free grammar** is a 4-tuple (V, Σ, R, S):

1. V is a finite set of *variables*.
2. Σ ($\Sigma \cap V = \emptyset$) is a finite set of *terminals*.
3. R is a finite set of *rules*, each of the form $A \to w$, where $A \in V$ and $w \in (V \cup \Sigma)^*$.
4. $S \in V$ is the *start* symbol.

- If $A \to w$ is a rule, then uAv *yields* uwv, written as $uAv \Rightarrow uwv$.
- We write $u \Rightarrow^* v$ if $u = v$ or a sequence $u_1, u_2, \ldots, u_k$ ($k \geq 0$) exists such that $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$.
- The *language of the grammar* is $\{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

Example CFGs

- $G_3 = (\{S\}, \{(,)\}, R, S)$, where R contains

$$S \to (S) \mid SS \mid \varepsilon.$$

  $L(G_3)$ is the language of all strings of properly nested parentheses such as $()(())$.

- $G_4 = (\{\langle\text{EXPR}\rangle, \langle\text{TERM}\rangle, \langle\text{FACTOR}\rangle\}, \{\mathtt{a}, +, \times, (,)\}, R, \langle\text{EXPR}\rangle)$, where R contains

$$\begin{array}{rcl} \langle\text{EXPR}\rangle & \to & \langle\text{EXPR}\rangle + \langle\text{TERM}\rangle \mid \langle\text{TERM}\rangle \\ \langle\text{TERM}\rangle & \to & \langle\text{TERM}\rangle \times \langle\text{FACTOR}\rangle \mid \langle\text{FACTOR}\rangle \\ \langle\text{FACTOR}\rangle & \to & (\langle\text{EXPR}\rangle) \mid \mathtt{a} \end{array}$$

  $L(G_4)$ is the language of algebraic expressions with the operations $+$ and $\times$ and a constant a such as $(a + a) \times a$.

Example CFGs (cont.)

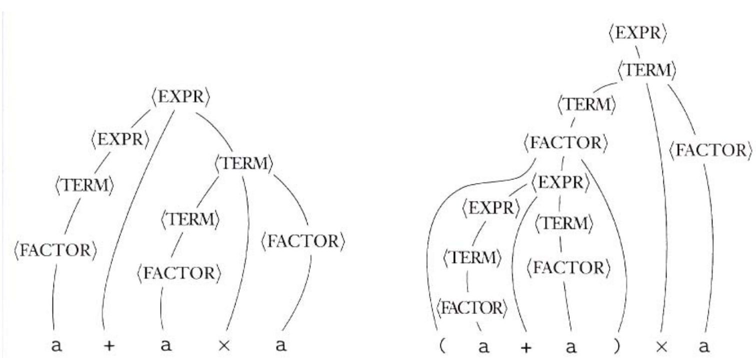

FIGURE 2.5
Parse trees for the strings a+a×a and (a+a)×a

Source: [Sipser 2006]

Designing CFGs

- If the CFL can be broken into simpler pieces, then break it and construct a grammar for each piece.
- If the CFL happens to be regular, then first construct a DFA and convert it into an equivalent CFG.
- Some CFLs contain strings with two substrings that correspond to each other in some way. Rules of the form $R \to uRv$ are useful for handling this situation.
- In more complex CFLs, the strings may contain certain structures that appear recursively as part of other structures. To achieve this effect, place the variable generating the structure in the location of the rules corresponding to where that structure may recursively appear.

From DFAs to CFGs

- Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$, we can construct a CFG $G = (V, \Sigma, R, S)$ as follows such that $L(G) = L(A)$.
- Make a variable R_i for each state $q_i \in Q$.
- Add the rule $R_i \to aR_j$ if $\delta(q_i, a) = q_j$.
- Add the rule $R_i \to \varepsilon$ if $q_i \in F$.
- Make R_0 (which corresponds to q_0) the start symbol.

Ambiguity

- Consider another grammar G_5 for algebraic expressions:

$$\begin{array}{rcl} \langle\text{EXPR}\rangle & \to & \langle\text{EXPR}\rangle + \langle\text{EXPR}\rangle \mid \\ & & \langle\text{EXPR}\rangle \times \langle\text{EXPR}\rangle \mid \\ & & (\langle\text{EXPR}\rangle) \mid \mathtt{a} \end{array}$$

- G_5 generates the string $\mathtt{a} + \mathtt{a} \times \mathtt{a}$ in two different ways.

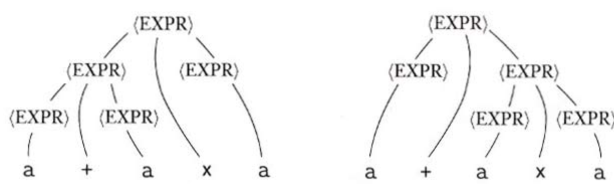

FIGURE 2.6
The two parse trees for the string a+a×a in grammar G_5

Source: [Sipser 2006]

Ambiguity (cont.)

- A derivation of a string in a grammar is a *leftmost derivation* if at every step the leftmost remaining variable is the one replaced.
- A parse tree represents one unique leftmost derivation.

Definition 2 (2.7)**.** A string is derived *ambiguously* in a grammar if it has two or more different leftmost derivations (or parse trees). A grammar is *ambiguous* if it generates some string ambiguously.

Chomsky Normal Form

- When working with context-free grammars, it is often convenient to have them in simplified form.

Definition 3 (2.8)**.** A context-free grammar is in **Chomsky normal form** if every rule is of the form

$$\begin{array}{rcl} A & \rightarrow & BC \quad \text{or} \\ A & \rightarrow & a \end{array}$$

where a is any terminal and B and C are not the start variable.

In addition,

$$S \rightarrow \varepsilon$$

is permitted if S is the start variable.

Chomsky Normal Form (cont.)

Theorem 4 (2.9)**.** *Any context-free language is generated by a context-free grammar in the Chomsky normal form.*

1. Add $S_0 \rightarrow S$, where S_0 is a new start symbol and S was the original start symbol.
2. Remove an ε rule $A \rightarrow \varepsilon$ if A is not the start symbol and add $R \rightarrow uv$ for each $R \rightarrow uAv$. $R \rightarrow \varepsilon$ is added unless it had been removed before. Repeat until no ε rule is left.
3. Remove a unit rule $A \rightarrow B$ and, for each $B \rightarrow u$, add $A \rightarrow u$ unless this is a unit rule previously removed. Repeat until no unit rule is left.
4. Replace each $A \rightarrow u_1u_2 \ldots u_k$ ($k \geq 3$) with $A \rightarrow u_1A_1$, $A_1 \rightarrow u_2A_2$, …, $A_{k-2} \rightarrow u_{k-1}u_k$. If u_i is a terminal, replace u_i with a new variable U_i and add $U_i \rightarrow u_i$.

An Example Conversion

Let us apply the described procedure to covert the following CFG to Chomsky normal form.

$$\begin{array}{rcl} S & \rightarrow & ASA \mid aB \\ A & \rightarrow & B \mid S \\ B & \rightarrow & b \mid \varepsilon \end{array}$$

- Add a new start symbol.

$$\begin{array}{rcl} S_0 & \rightarrow & S \\ S & \rightarrow & ASA \mid aB \\ A & \rightarrow & B \mid S \\ B & \rightarrow & b \mid \varepsilon \end{array}$$

An Example Conversion (cont.)

- Remove ε rule $B \rightarrow \varepsilon$.

$$\begin{array}{rcl} S_0 & \rightarrow & S \\ S & \rightarrow & ASA \mid aB \mid a \\ A & \rightarrow & B \mid S \mid \varepsilon \\ B & \rightarrow & b \cancel{\mid \varepsilon} \end{array}$$

- Remove $A \rightarrow \varepsilon$.

$$\begin{array}{rcl} S_0 & \rightarrow & S \\ S & \rightarrow & ASA \mid aB \mid a \mid SA \mid AS \mid S \\ A & \rightarrow & B \mid S \cancel{\mid \varepsilon} \\ B & \rightarrow & b \end{array}$$

An Example Conversion (cont.)

- Remove unit rule $S \to S$.

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

- Remove $S_0 \to S$.

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

An Example Conversion (cont.)

- Remove $A \to B$.

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

- Remove $A \to S$.

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

An Example Conversion (cont.)

- Convert $S_0 \to ASA$, $S \to ASA$, and $A \to ASA$.

$$\begin{array}{rcl} S_0 & \to & AA_{1,1} \mid aB \mid a \mid SA \mid AS \\ S & \to & AA_{2,1} \mid aB \mid a \mid SA \mid AS \\ A & \to & b \mid AA_{3,1} \mid aB \mid a \mid SA \mid AS \\ A_{1,1} & \to & SA \\ A_{2,1} & \to & SA \\ A_{3,1} & \to & SA \\ B & \to & b \end{array}$$

An Example Conversion (cont.)

- Convert $S_0 \to aB$, $S \to aB$, and $A \to aB$.

$$\begin{array}{rcl} S_0 & \to & AA_{1,1} \mid U_1B \mid a \mid SA \mid AS \\ S & \to & AA_{2,1} \mid U_2B \mid a \mid SA \mid AS \\ A & \to & b \mid AA_{3,1} \mid U_3B \mid a \mid SA \mid AS \\ A_{1,1} & \to & SA \\ A_{2,1} & \to & SA \\ A_{3,1} & \to & SA \\ U_1 & \to & a \\ U_2 & \to & a \\ U_3 & \to & a \\ B & \to & b \end{array}$$

2 Pushdown Automata

Pushdown Automata

- *Pushdown automata* (PDAs) are like nondeterministic finite automata but have an extra component called a *stack*.
- A stack is valuable because it can hold an *unlimited* amount of information.
- In contrast with the finite automata situation, *nondeterminism* adds power to the capability that pushdown automata would have if they were allowed only to be deterministic.
- Pushdown automata are equivalent in power to context-free grammars.
- To prove that a language is context-free, we can give either a context-free grammar *generating* it or a pushdown automaton *recognizing* it.

Pushdown Automata (cont.)

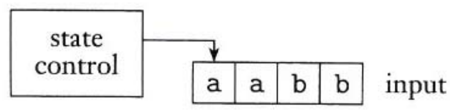

FIGURE 2.11
Schematic of a finite automaton

Source: [Sipser 2006]

Pushdown Automata (cont.)

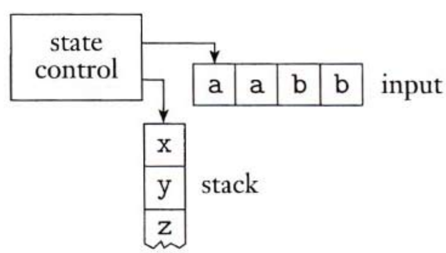

FIGURE 2.12
Schematic of a pushdown automaton

Source: [Sipser 2006]

Definition of a PDA

Definition 5 (2.13)**.** 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. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

An Example PDA

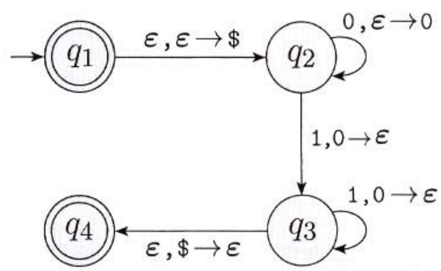

FIGURE **2.15**
State diagram for the PDA M_1 that recognizes $\{0^n 1^n \mid n \geq 0\}$

Source: [Sipser 2006]

Computation of a PDA

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA and w be a string over Σ.
- We say that M *accepts* w if we can write $w = w_1 w_2 \dots w_n$, where $w_i \in \Sigma_\varepsilon$, and sequences of states $r_0, r_1, \dots, r_n \in Q$ and strings $s_0, s_1, \dots, s_n \in \Gamma^*$ exist such that:
  1. $r_0 = q_0$ and $s_0 = \varepsilon$,
  2. for $i = 0, 1, \dots, n-1$, $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ and $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$.
  3. $r_n \in F$.

Computation of a PDA (cont.)

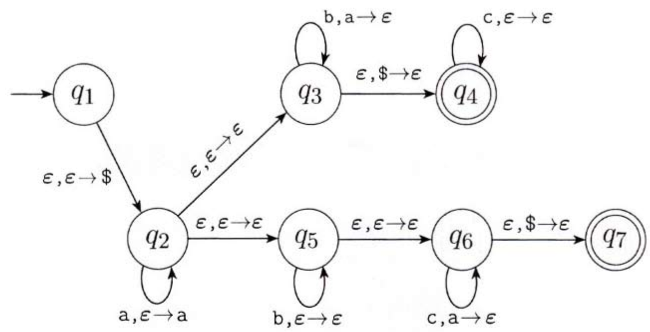

FIGURE 2.17
State diagram for PDA M_2 that recognizes
$\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

Source: [Sipser 2006]

Computation of a PDA (cont.)

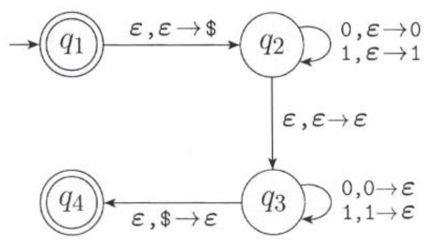

FIGURE 2.19
State diagram for the PDA M_3 that recognizes $\{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$

Source: [Sipser 2006]

Equivalence of PDAs and CFGs

Theorem 6 (2.20)**.** *A language is context free if and only if some pushdown automaton recognizes it.*

- Recall that a context-free language is one that can be described with a context-free grammar.
- We show how to convert any context-free grammar into a pushdown automaton that recognizes the same language and vice versa.

CFGs $\subseteq$ PDAs

Lemma 7 (2.21)**.** *If a language is context free, then some pushdown automaton recognizes it.*

- Let G be a CFG generating language A. We convert G into a PDA P that recognizes A.
- P begins by writing the start variable on its stack.
- P's nondeterminism allows it to guess the sequence of correct substitutions. For example, to simulate that $A \to u$ is selected, A on the top of the stack is replaced with u.
- The top symbol on the stack may not be a variable. Any terminal symbols appearing before the first variable are matched immediately with symbols in the input string.

CFGs $\subseteq$ PDAs (cont.)

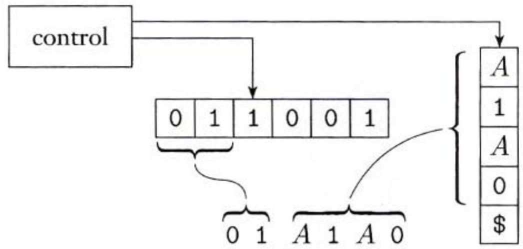

FIGURE 2.22
P representing the intermediate string $01A1A0$

Source: [Sipser 2006]

CFGs $\subseteq$ PDAs (cont.)

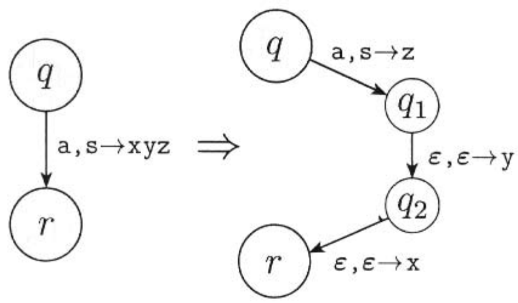

FIGURE 2.23
Implementing the shorthand $(r, xyz) \in \delta(q, a, s)$

Source: [Sipser 2006]

CFGs $\subseteq$ PDAs (cont.)

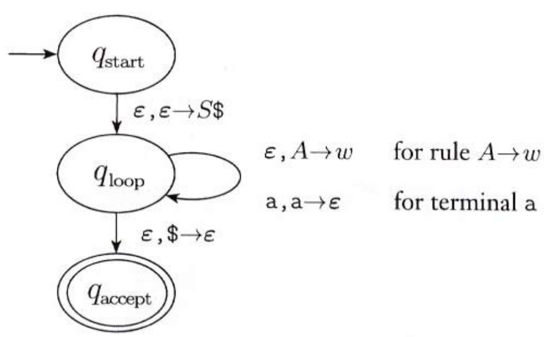

FIGURE 2.24
State diagram of P

Source: [Sipser 2006]

CFGs $\subseteq$ PDAs (cont.)

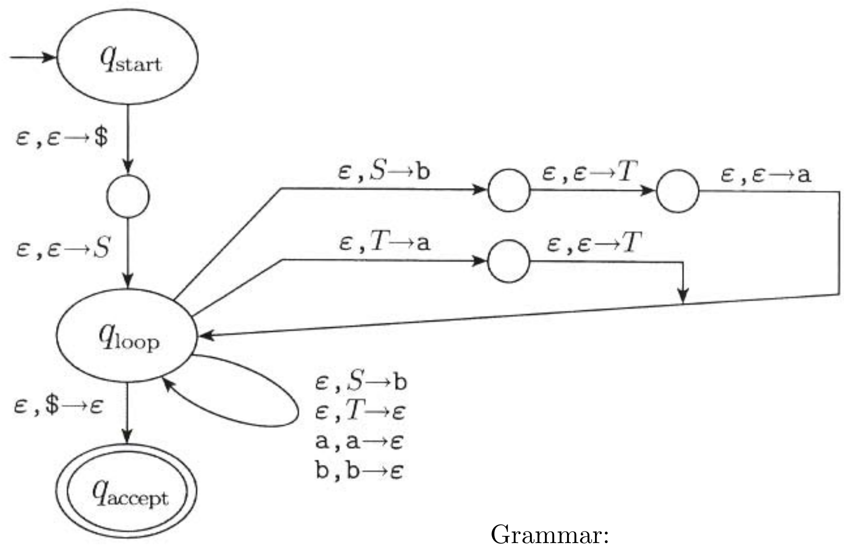

FIGURE 2.26
State diagram of P_1

Grammar:

$$\begin{array}{lcl} S & \to & aTb \mid b \\ T & \to & Ta \mid \epsilon \end{array}$$

Source: [Sipser 2006]

PDAs $\subseteq$ CFGs

Lemma 8 (2.27)**.** *If some pushdown automaton recognizes a language, then it is context free.*

- Convert a PDA P into an equivalent CFG G.
- Modify P so that
  1. it has a single accept state,
  2. it empties its stack before accepting, and
  3. each transition either pushes a symbol onto the stack or pops one off the stack, but not both.

PDAs $\subseteq$ CFGs (cont.)

- For each pair of states p and q in P, grammar G will have a variable A_{pq}.
- A_{pq} generates all the strings that can take P from p with an empty stack to q with an empty stack (or without touching the contents already on the stack when P was in state p).
- The start symbol is $A_{q_0 q_a}$, where q_0 is the initial state and q_a the only accept state of P.
- Add $A_{pq} \to aA_{rs}b$ to G if $\delta(p, a, \varepsilon)$ contains (r, t) and $\delta(s, b, t)$ contains (q, ε).
- Add $A_{pq} \to A_{pr}A_{rq}$ to G for each $p, q, r \in Q$.
- Add $A_{pp} \to \varepsilon$ to G for each $p \in Q$.

PDAs ⊆ CFGs (cont.)

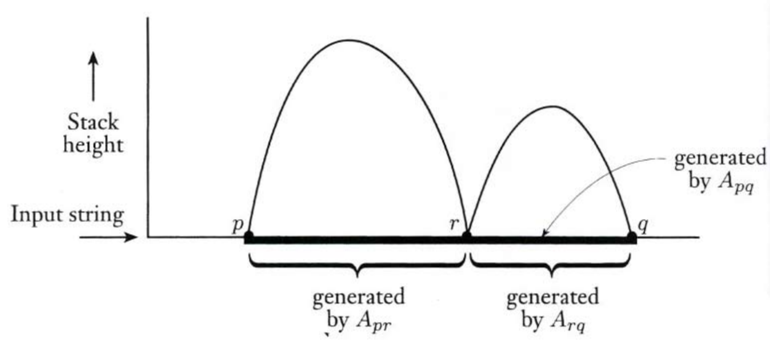

FIGURE 2.28
PDA computation corresponding to the rule $A_{pq} \to A_{pr}A_{rq}$

Source: [Sipser 2006]

PDAs ⊆ CFGs (cont.)

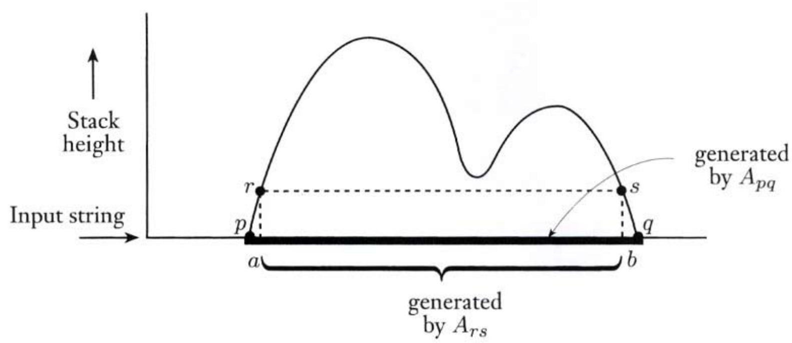

FIGURE 2.29
PDA computation corresponding to the rule $A_{pq} \to aA_{rs}b$

Source: [Sipser 2006]

PDAs ⊆ CFGs (cont.)

Claim 1 (2.30)**.** *If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack.*

Claim 2 (2.31)**.** *If x can bring P from p with empty stack to q with empty stack, then A_{pq} generates x.*

Regular vs. Context-Free Languages

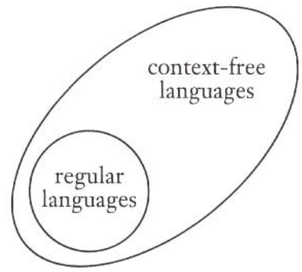

FIGURE 2.33
Relationship of the regular and context-free languages

Source: [Sipser 2006]

Note: this is an inclusion relationship between two classes, not two specific languages (e.g., $\{0^n1^n \mid n \geq 0\} \subseteq L(0^*1^*)$).

3 Pumping Lemma

The Pumping Lemma for CFL

Theorem 9 (2.34)**.** *If A is a context-free language, then there is a number p such that, if s is a string in A and $|s| \geq p$, then s may be divided into five pieces, $s = uvxyz$, satisfying the conditions:*

1. *for each $i \geq 0$, $uv^ixy^iz \in A$,*
2. *$|vy| > 0$, and*
3. *$|vxy| \leq p$.*

- Let G be a CFG that generates A.
- Consider a "sufficiently long" string s in A that satisfies the following condition:

  The parse tree for s is very tall so as to have a long path on which some variable symbol R of G repeats.

The Pumping Lemma for CFL (cont.)

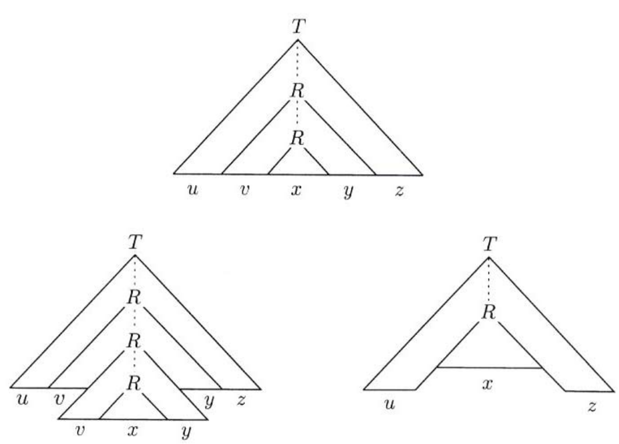

FIGURE 2.35
Surgery on parse trees

Source: [Sipser 2006]

The Pumping Lemma for CFL (cont.)

- Let b be the upper bound on the length of w for any production rule $A \to w$ in G.
- Take p to be $b^{|V|+1}$, where V is the set of variables of G. A string of length at least p is sufficiently long.
- Consider the *smallest* parse tree of a string s whose length is at least $b^{|V|+1}$.
  - vy cannot be empty, otherwise we would have an even smaller parse tree.
  - To ensure $|vxy| \leq p$, choose an R that occurs twice within the bottom $|V| + 1$ levels of a path.

Example Non-Context-Free Languages
$B = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free.

- Let s be $a^p b^p c^p$, where p is the pumping length.
- Cases of dividing s as $uvxyz$ (where $|vy| > 0$ and $|vxy| \leq p$):
  1. Both v and y contain only one type of symbol, e.g.,
     $a \cdot \underbrace{\cdots}_{v} \underbrace{\cdot a\, b \cdot}_{x} \underbrace{\cdots}_{y} \cdot b\, \overbrace{c \cdots c}^{p}$, in which case, uv^2xy^2z will have more a's or b's than c's and so is not in B.
  2. Either v or y contains more than one type of symbol, e.g.,
     $a \cdot \underbrace{\cdots}_{v} \underbrace{\cdots}_{x} \underbrace{\cdot ab \cdot}_{y} \cdots bc \cdots c$, in which case, uv^2xy^2z will have some a's and b's out of order and so is not in B.

Example Non-Context-Free Languages (cont.)
$C = \{a^i b^j c^k \mid 0 \leq i \leq j \leq k\}$ is not context-free.

- Let s be $a^p b^p c^p$.
- Cases of dividing s as $uvxyz$ (where $|vy| > 0$ and $|vxy| \leq p$):
  1. Both v and y contain only one type of symbol, e.g.,
     $a \cdot \underbrace{\cdots}_{v} \underbrace{\cdot a\, b \cdot}_{x} \underbrace{\cdots}_{y} \cdot b\, \overbrace{c \cdots c}^{p}$, in which case, uv^2xy^2z will have more a's or b's than c's and so is not in C, or
     $a \cdots ab \cdot \underbrace{\cdots}_{v} \underbrace{\cdot bc \cdot}_{x} \underbrace{\cdots}_{y} \cdot c$, in which case, uv^0xy^0z will have less b's or c's than a's and so is not in C.
  2. Either v or y contains more than one type of symbol, e.g.,
     $a \cdot \underbrace{\cdots}_{v} \underbrace{\cdots}_{x} \underbrace{\cdot ab \cdot}_{y} \cdots bc \cdots c$, in which case, uv^2xy^2z will have some a's and b's out of order and so is not in C.

Example Non-Context-Free Languages (cont.)

$D = \{ww \mid w \in \{0,1\}^*\}$ is not context-free.

- Let s be $0^p 1^p 0^p 1^p$.
- Cases of dividing s as $uvxyz$ (where $|vy| > 0$ and $|vxy| \le p$):
  1. The substring vxy is entirely within the first or second half, e.g.,
     $\overbrace{0\cdots\cdots 0}^{p}\overbrace{1\cdots\cdots 1}^{p}\overbrace{0\cdots 0}^{p}\overbrace{1\cdots 1}^{p}$ (with $\underbrace{\cdots 0 1 \cdots}_{vxy}$), in which case, uv^2xy^2z will move a 1 to the first position of the second half and so is not of the form ww.
  2. The substring vxy straddles the midpoint of s, i.e.,
     $0\cdots 01 \cdot \underbrace{\cdots 10 \cdots}_{vxy} \cdot 01 \cdots 1$, in which case, uv^0xy^0z will have the form $0^p 1^i 0^j 1^p$ with either i or j less than p and so is not of the form ww.