T065001: Introduction to Formal Languages

Lecture 7: Pushdown automata, context-free languages, grammars (2)

Chapter 2.2 in Sipser's textbook

2025-06-02

(Lecture slides by Yih-Kuen Tsay)

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.)

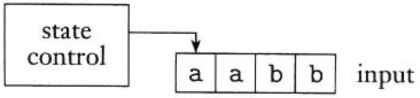


FIGURE 2.11
Schematic of a finite automaton

Pushdown Automata (cont.)

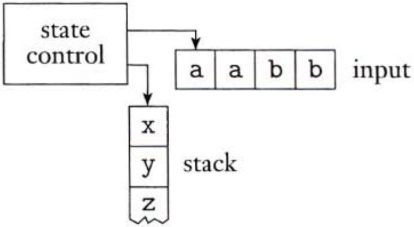


FIGURE 2.12
Schematic of a pushdown automaton

(From Chapter 1.2)

- The transition function of an NFA takes a state and an input symbol *or the empty string* and produces *a set of possible next states*.
- Let $\mathcal{P}(Q)$ be the power set of Q and let Σ_ε denote $\Sigma \cup \{\varepsilon\}$.

Example: Let $Q = \{q_0, q_1, q_2\}$. Then
$\mathcal{P}(Q) = \big\{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\big\}$.

(From Chapter 1.2)

- The transition function of an NFA takes a state and an input symbol *or the empty string* and produces *a set of possible next states*.
- Let $\mathcal{P}(Q)$ be the power set of Q and let Σ_ε denote $\Sigma \cup \{\varepsilon\}$.

Definition (1.37)

A *nondeterministic finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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

Definition of a PDA

How is the PDA model similar to the NFA model in Chapter 1.2?

Same basic idea: Read one symbol of the input at a time, follow the arrows in the state diagram, and accept if we end up in an accept state.

Definition of a PDA

How is the PDA model similar to the NFA model in Chapter 1.2?

Same basic idea: Read one symbol of the input at a time, follow the arrows in the state diagram, and accept if we end up in an accept state.

Nondeterminism $\Rightarrow$ There can be several legal next moves for a PDA. Consequently, the values of a PDA's transition function will also be sets.

Definition of a PDA

How is the PDA model similar to the NFA model in Chapter 1.2?

Same basic idea: Read one symbol of the input at a time, follow the arrows in the state diagram, and accept if we end up in an accept state.

Nondeterminism $\Rightarrow$ There can be several legal next moves for a PDA. Consequently, the values of a PDA's transition function will also be sets.

How do the PDA model and the NFA model in Chapter 1.2 differ?

The current state, the read input symbol, and the top symbol on the stack determine the next move for a PDA.

Definition of a PDA

How is the PDA model similar to the NFA model in Chapter 1.2?

Same basic idea: Read one symbol of the input at a time, follow the arrows in the state diagram, and accept if we end up in an accept state.

Nondeterminism $\Rightarrow$ There can be several legal next moves for a PDA. Consequently, the values of a PDA's transition function will also be sets.

How do the PDA model and the NFA model in Chapter 1.2 differ?

The current state, the read input symbol, and the top symbol on the stack determine the next move for a PDA.

In each move, the PDA can enter a new state and possibly change the symbol at the top of the stack.

Definition of a PDA

How is the PDA model similar to the NFA model in Chapter 1.2?

Same basic idea: Read one symbol of the input at a time, follow the arrows in the state diagram, and accept if we end up in an accept state.

Nondeterminism $\Rightarrow$ There can be several legal next moves for a PDA. Consequently, the values of a PDA's transition function will also be sets.

How do the PDA model and the NFA model in Chapter 1.2 differ?

The current state, the read input symbol, and the top symbol on the stack determine the next move for a PDA.

In each move, the PDA can enter a new state and possibly change the symbol at the top of the stack.

Furthermore, the stack may use a different alphabet than the input.

Definition of a PDA

Definition (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.

where $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$.

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 $w = w_1 w_2 \ldots w_n$, where $w_i \in \Sigma_\varepsilon$, and sequences of states $r_0, r_1, \ldots, r_n \in Q$ and strings $s_0, s_1, \ldots, s_n \in \Gamma^*$ exist such that:
  1. $r_0 = q_0$ and $s_0 = \varepsilon$,
  2. for $i = 0, 1, \ldots, n-1$, $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ 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

- 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. $r_n \in F$.

Observe that the accept states take effect only when the PDA is at the end of the input, i.e., M must read the entire string w.

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 $w = w_1 w_2 \ldots w_n$, where $w_i \in \Sigma_\varepsilon$, and sequences of states $r_0, r_1, \ldots, r_n \in Q$ and strings $s_0, s_1, \ldots, s_n \in \Gamma^*$ exist such that:
  1. $r_0 = q_0$ and $s_0 = \varepsilon$,
  2. for $i = 0, 1, \ldots, n-1$, $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ and $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$.
  3. $r_n \in F$.

Observe that the accept states take effect only when the PDA is at the end of the input, i.e., M must read the entire string w.

Notation: The set of all strings that are accepted by a PDA M is denoted by $L(M)$. (We also say that M *recognizes* this language.)

Pushdown Automata (cont.)

Notation used in state diagrams:

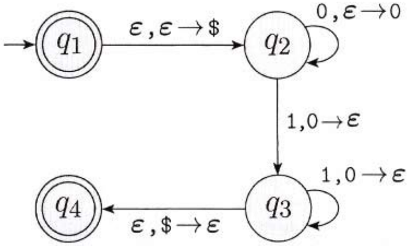
$$\begin{tikzcd} q_i \arrow[r, "\texttt{a,b} \to \texttt{c}"] & q_j \end{tikzcd}$$

A label of the form "$\texttt{a,b} \to \texttt{c}$" on a transition from a state q_i to a state q_j means that $(q_j, \texttt{c}) \in \delta(q_i, \texttt{a}, \texttt{b})$, i.e.:

> *If the machine is in state q_i, it reads the symbol* `a` *from the input, and the symbol* `b` *is at the top of the stack, then it can replace the* `b` *at the top of the stack by the symbol* `c` *and go to state q_j.*

(Any of $\texttt{a}, \texttt{b}, \texttt{c}$ may be ε.)

Example 1



q_1
$\varepsilon,\varepsilon\rightarrow\$$
q_2
$0,\varepsilon\rightarrow 0$
$1,0\rightarrow\varepsilon$
q_3
$1,0\rightarrow\varepsilon$
q_4
$\varepsilon,\$\rightarrow\varepsilon$

FIGURE **2.15**

Example 1

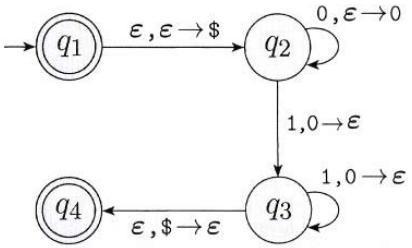


FIGURE 2.15
State diagram for the PDA M_1 that recognizes $\{0^n1^n \mid n \geq 0\}$

Example 1

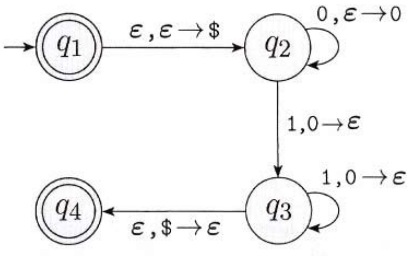
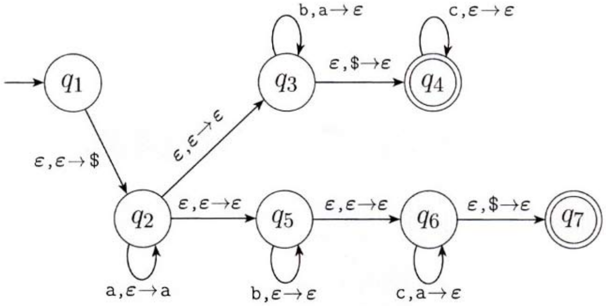


FIGURE 2.15
State diagram for the PDA M_1 that recognizes $\{0^n 1^n \mid n \geq 0\}$

Remark: Marking the bottom of the stack with a special symbol (like \$ above) is a useful trick that allows a PDA to test for an empty stack.

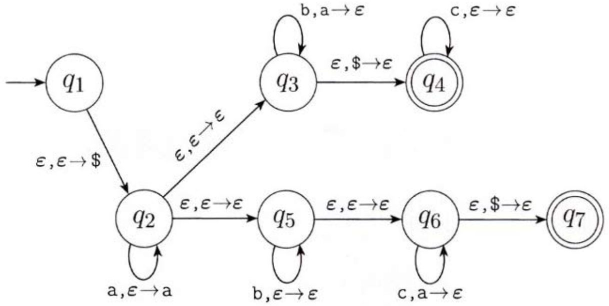
Example 2



q_1
q_2
q_3
q_4
q_5
q_6
q_7
$\varepsilon,\varepsilon\to\$$
$\varepsilon,\varepsilon\to\varepsilon$
$\text{b},\text{a}\to\varepsilon$
$\varepsilon,\$\to\varepsilon$
$\text{c},\varepsilon\to\varepsilon$
$\text{a},\varepsilon\to\text{a}$
$\varepsilon,\varepsilon\to\varepsilon$
$\text{b},\varepsilon\to\varepsilon$
$\varepsilon,\varepsilon\to\varepsilon$
$\text{c},\text{a}\to\varepsilon$
$\varepsilon,\$\to\varepsilon$

FIGURE **2.17**

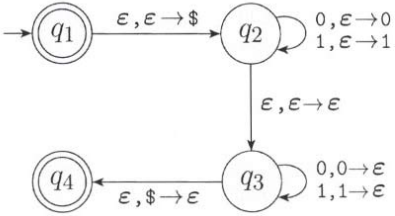
Example 2



q_1
q_2
q_3
q_4
q_5
q_6
q_7
$\varepsilon,\varepsilon\to\$$
$\varepsilon,\varepsilon\to\varepsilon$
b,a$\to\varepsilon$
$\varepsilon,\$\to\varepsilon$
c,$\varepsilon\to\varepsilon$
a,$\varepsilon\to$a
$\varepsilon,\varepsilon\to\varepsilon$
b,$\varepsilon\to\varepsilon$
$\varepsilon,\varepsilon\to\varepsilon$
c,a$\to\varepsilon$
$\varepsilon,\$\to\varepsilon$

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

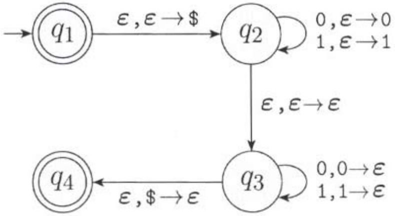
Example 3



q_1
$\varepsilon, \varepsilon \to \$$
q_2
$0, \varepsilon \to 0$
$1, \varepsilon \to 1$
$\varepsilon, \varepsilon \to \varepsilon$
q_3
$0, 0 \to \varepsilon$
$1, 1 \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
q_4

FIGURE **2.19**

Example 3



q_1
$\varepsilon,\varepsilon\to\$$
q_2
$0,\varepsilon\to 0$
$1,\varepsilon\to 1$
$\varepsilon,\varepsilon\to\varepsilon$
q_3
$0,0\to\varepsilon$
$1,1\to\varepsilon$
$\varepsilon,\$\to\varepsilon$
q_4

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

Equivalence of PDAs and CFGs

Theorem (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.

Equivalence of PDAs and CFGs

To simplify the explanation, we extend the notation for transitions so that we can push more than one symbol onto the stack in one step as follows:

$(r, u) \in \delta(q, a, s)$, where u can be a string and not just a single symbol.

Equivalence of PDAs and CFGs

To simplify the explanation, we extend the notation for transitions so that we can push more than one symbol onto the stack in one step as follows:

$(r, u) \in \delta(q, a, s)$, where u can be a string and not just a single symbol.

Interpretation:

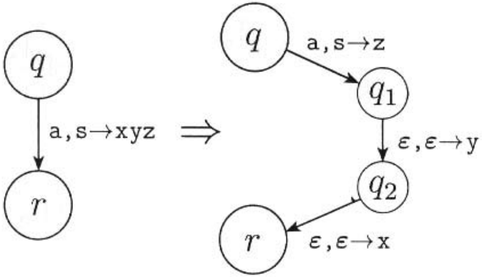


FIGURE **2.23**
Implementing the shorthand $(r, xyz) \in \delta(q, a, s)$

CFGs $\subseteq$ PDAs

Lemma (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.
  For any w, P will accept input w if and only if G can generate w.

CFGs $\subseteq$ PDAs

Lemma (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.
  For any w, P will accept input w if and only if G can generate w.
- 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.
- To ensure that the top symbol on the stack is a variable, any terminal symbols appearing before the first variable are matched immediately with symbols in the input string.

CFGs $\subseteq$ PDAs

Lemma (2.21)

If a language is context free, then some pushdown automaton recognizes it.

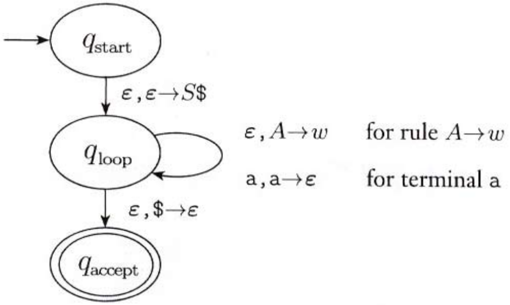
- Let G be a CFG generating language A. We convert G into a PDA P that recognizes A.
  For any w, P will accept input w if and only if G can generate w.
- P begins by writing the start variable on its stack.
- P's nondeterminism allows it to guess the sequence of correct substitutions. For example, to simulate that $A \rightarrow u$ is selected, A on the top of the stack is replaced with u.
- To ensure that the top symbol on the stack is a variable, any terminal symbols appearing before the first variable are matched immediately with symbols in the input string.

Note that P avoids having to keep the entire derived string on the stack.

CFGs $\subseteq$ PDAs (cont.)

The resulting PDA will have the form shown below.

Many transitions from q_{loop} to q_{loop}! Each one handles one case where the top of the stack is either: (i) a variable; or (ii) a terminal.



q_{start}
$\varepsilon, \varepsilon \to S\$$
q_{loop}
$\varepsilon, A \to w$ for rule $A \to w$
$\text{a}, \text{a} \to \varepsilon$ for terminal a
$\varepsilon, \$ \to \varepsilon$
q_{accept}

FIGURE 2.24
State diagram of P

CFGs $\subseteq$ PDAs (cont.)

The states of P are $\{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}\} \cup E$, where E is the set of states needed to implement the shorthand notation in Figure 2.23.

How P operates:

1. Place the marker symbol $ and the start variable on the stack.

2. Repeat the following steps forever.

- **a.** If the top of stack is a variable symbol A, nondeterministically select one of the rules for A and substitute A by the string on the right-hand side of the rule.
- **b.** If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If they match, repeat. If they do not match, reject on this branch of the nondeterminism.
- **c.** If the top of stack is the symbol $, enter the accept state. Doing so accepts the input if it has all been read.

CFGs $\subseteq$ PDAs (cont.)

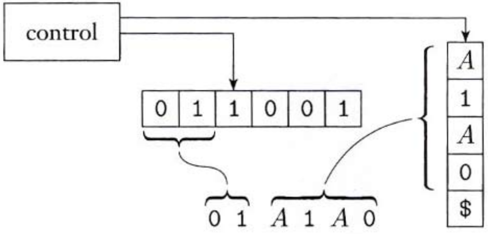


FIGURE **2.22**
P representing the intermediate string $01A1A0$

Example: Convert a CFG to a PDA using Lemma 2.21

Suppose we want to convert the following CFG into an equivalent PDA:

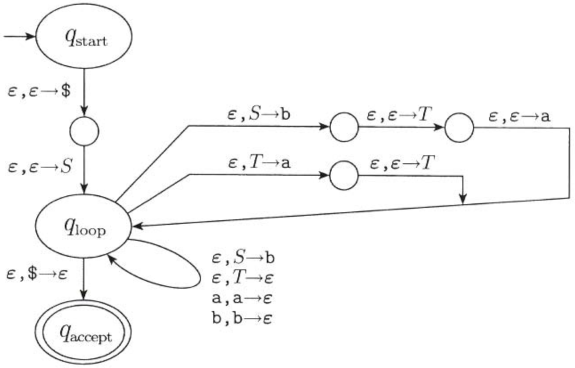
$$\begin{array}{rcl} S & \rightarrow & aTb \mid b \\ T & \rightarrow & Ta \mid \epsilon \end{array}$$

Example: Convert a CFG to a PDA using Lemma 2.21

Suppose we want to convert the following CFG into an equivalent PDA:

$$\begin{aligned} S &\rightarrow aTb \mid b \\ T &\rightarrow Ta \mid \epsilon \end{aligned}$$

Applying the construction in the proof of Lemma 2.21 gives:



q_{start}
$\varepsilon, \varepsilon \rightarrow \$$
$\varepsilon, \varepsilon \rightarrow S$
$\varepsilon, S \rightarrow \text{b}$
$\varepsilon, \varepsilon \rightarrow T$
$\varepsilon, \varepsilon \rightarrow \text{a}$
$\varepsilon, T \rightarrow \text{a}$
$\varepsilon, \varepsilon \rightarrow T$
q_{loop}
$\varepsilon, S \rightarrow \text{b}$
$\varepsilon, T \rightarrow \varepsilon$
$\text{a}, \text{a} \rightarrow \varepsilon$
$\text{b}, \text{b} \rightarrow \varepsilon$
$\varepsilon, \$ \rightarrow \varepsilon$
q_{accept}

Equivalence of PDAs and CFGs

Theorem (2.20)

A language is context free if and only if some pushdown automaton recognizes it.

Lemma 2.21 established one direction (CFG $\longrightarrow$ PDA).

Equivalence of PDAs and CFGs

Theorem (2.20)

A language is context free if and only if some pushdown automaton recognizes it.

Lemma 2.21 established one direction (CFG $\rightarrow$ PDA).

To complete the proof of Theorem 2.20, we need to show the other direction (PDA $\rightarrow$ CFG), too.

PDAs $\subseteq$ CFGs

Lemma (2.27)

If some pushdown automaton recognizes a language, then it is context free.

- Convert a PDA P into an equivalent CFG G.
  (We need to construct G so that G generates a string if that string causes P to go from its start state to an accept state.)

PDAs $\subseteq$ CFGs

Lemma (2.27)

If some pushdown automaton recognizes a language, then it is context free.

- Convert a PDA P into an equivalent CFG G.
  (We need to construct G so that G generates a string if that string causes P to go from its start state to an accept state.)
- To simplify the task, first 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

Lemma (2.27)

If some pushdown automaton recognizes a language, then it is context free.

- Convert a PDA P into an equivalent CFG G.
  (We need to construct G so that G generates a string if that string causes P to go from its start state to an accept state.)
- To simplify the task, first 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.

  $\Rightarrow$ For any string x, P's first move on x must be a push and the last move on x must be a pop.

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.

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.
- For each $p, q, r \in Q$:
  Add $A_{pq} \rightarrow A_{pr} A_{rq}$ to G.

PDAs $\subseteq$ CFGs (cont.)

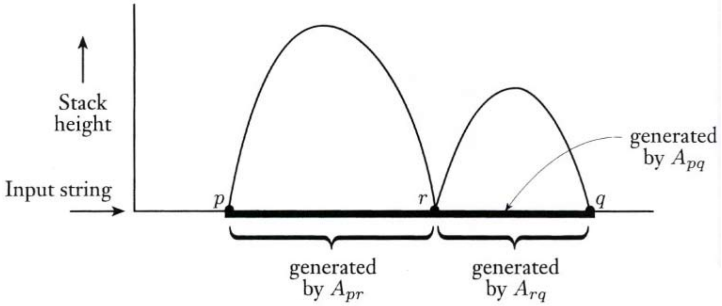


FIGURE 2.28
PDA computation corresponding to the rule $A_{pq} \rightarrow A_{pr}A_{rq}$

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.
- For each $p, q, r \in Q$:
  Add $A_{pq} \rightarrow A_{pr} A_{rq}$ to G.
- For each $p, q, r, s \in Q$, $t \in \Gamma$, and $a, b \in \Sigma_\varepsilon$:
  Add $A_{pq} \rightarrow a A_{rs} b$ to G if $(r, t) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, t)$.

PDAs $\subseteq$ CFGs (cont.)

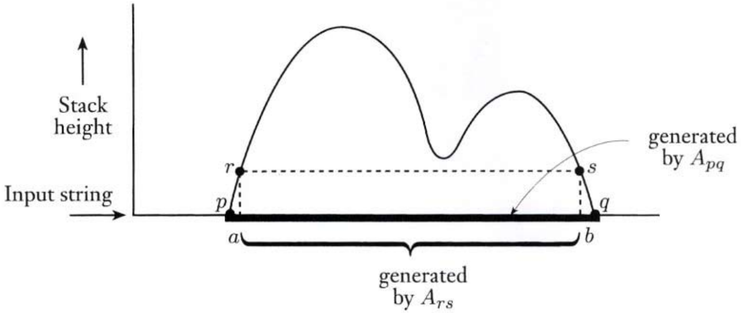


FIGURE 2.29
PDA computation corresponding to the rule $A_{pq} \to aA_{rs}b$

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.
- For each $p, q, r \in Q$:
  Add $A_{pq} \rightarrow A_{pr} A_{rq}$ to G.
- For each $p, q, r, s \in Q$, $t \in \Gamma$, and $a, b \in \Sigma_\varepsilon$:
  Add $A_{pq} \rightarrow a A_{rs} b$ to G if $(r, t) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, t)$.
- For each $p \in Q$:
  Add $A_{pp} \rightarrow \varepsilon$ to G.

PDAs $\subseteq$ CFGs (cont.)

Claim (2.30)

If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack.

Claim (2.31)

If x can bring P from p with empty stack to q with empty stack, then A_{pq} generates x.

PDAs $\subseteq$ CFGs (cont.)

Claim (2.30)

If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack.

Claim (2.31)

If x can bring P from p with empty stack to q with empty stack, then A_{pq} generates x.

Both claims can be proved by induction:

Claim 2.30: Induction on the number of steps in the derivation of x.

Claim 2.31: Induction on the number of steps in the computation of P on input x that goes from p to q with empty stacks.

Equivalence of PDAs and CFGs

Finally, combining Lemmas 2.21 and 2.27 gives:

Theorem (2.20)

A language is context free if and only if some pushdown automaton recognizes it.

Regular vs. Context-Free Languages

By definition, every regular language is recognized by a finite automaton.

A finite automaton is a special case of a pushdown automaton that ignores its stack, i.e., all of its transitions are of the form “$\mathtt{a,}\varepsilon \to \varepsilon$“.

$\Rightarrow$ Every regular language is a context-free language (but not the other way around!).

Regular vs. Context-Free Languages

By definition, every regular language is recognized by a finite automaton.
A finite automaton is a special case of a pushdown automaton that ignores its stack, i.e., all of its transitions are of the form “$\texttt{a,}\varepsilon \to \varepsilon$“.

$\Rightarrow$ Every regular language is a context-free language (but not the other way around!).

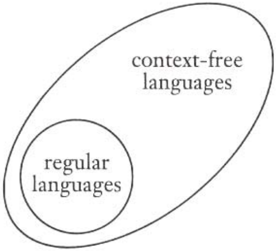


FIGURE **2.33**
Relationship of the regular and context-free languages

From DFAs to CFGs

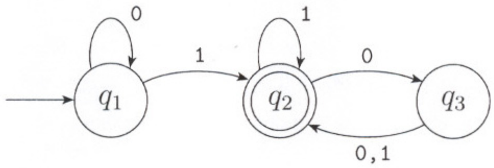
Remark: Here is another way to see that every regular language is a context-free language, without using the concept of PDAs:

From DFAs to CFGs

Remark: Here is another way to see that every regular language is a context-free language, without using the concept of PDAs:

- Given any DFA $A = (Q, \Sigma, \delta, q_1, F)$, we can construct a CFG $G = (V, \Sigma, R, S)$ with $L(G) = L(A)$ directly as follows.
- Make a variable R_i for each state $q_i \in Q$.
- Add the rule $R_i \rightarrow aR_j$ if $\delta(q_i, a) = q_j$.
- Add the rule $R_i \rightarrow \varepsilon$ if $q_i \in F$.
- Let R_1 (corresponding to A's start state q_1) be G's start symbol.

Example:



$$\Rightarrow \qquad \begin{aligned} R_1 &\rightarrow 0\,R_1 \mid 1\,R_2 \\ R_2 &\rightarrow 0\,R_3 \mid 1\,R_2 \mid \varepsilon \\ R_3 &\rightarrow 0\,R_2 \mid 1\,R_2 \end{aligned}$$