2 Context-Free Languages

(Part 2 of 2)

Yajun Yang

yjyang@tju.edu.cn

School of Computer Science and Technology

Tianjin University

2016

Outline

Context-Free Languages

regular languages 正则语言

- finite automata: DFA / NFA
- regular expressions

some simple languages, such as $\{0^n1^n \mid n \geq 0\}$, are ***not*** regular languages.

Context-Free Languages

regular languages 正则语言

- finite automata: DFA / NFA
- regular expressions

some simple languages, such as $\{0^n1^n \mid n \geq 0\}$, are ***not*** regular languages.

context-free languages 上下文无关语言

- pushdown automata 下推自动机

- first used in the study of human languages
- in the specification and compilation of programming languages
  - ***parser***
  - the construction of a parser from a context-free grammar

Outline

Outline

Pushdown Automata 下推自动机

Pushdown Automata (PDA): we introduce a new type of computational model.

- like NFA but have an extra component called a ***stack***.

Pushdown Automata 下推自动机

Pushdown Automata (PDA): we introduce a new type of computational model.

- like NFA but have an extra component called a ***stack***.
- the stack provides additional memory beyond the finite amount available in the control.

Pushdown Automata 下推自动机

Pushdown Automata (PDA): we introduce a new type of computational model.

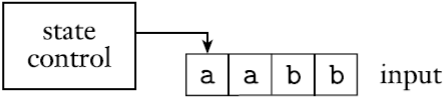
- like NFA but have an extra component called a ***stack***.
- the stack provides additional memory beyond the finite amount available in the control.
- the stack allows PDA to recognize some nonregular languages.

PDA are equivalent in power to CFG

- two options for proving that a language is context free
  - give either a CFG generating it (generator)
  - or a PDA recognizing it (recognizer)

Pushdown Automata 下推自动机

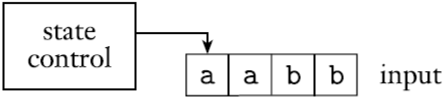
DFA/NFA vs. PDA



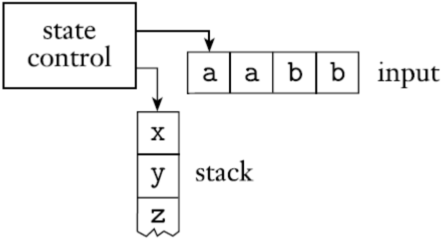
Schematic of DFA/NFA

Pushdown Automata 下推自动机

DFA/NFA vs. PDA



Schematic of DFA/NFA



Schematic of PDA

Pushdown Automata 下推自动机

PDA can write symbols on the stack and read them back later.

Writing a symbol ***"pushes down"*** all the other symbols on the stack.

- all access to the stack may be done only at the top: "last in, first out"
- ***pushing***: writing a symbol on the top of the stack
- ***popping***: removing a symbol on the top of the stack

A stack can hold an unlimited amount of information.

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

- a DFA/NFA is unable to recognize it.
- A PDA is able to recognize it.

Pushdown Automata 下推自动机

Deterministic and nondeterministic PDA are ***not*** equivalent in power.

- Nondeterministic PDA recognize certain languages that ***no*** deterministic PDA can recognize.
- Recall that DFA and NFA do recognize the same class of languages.
- So the pushdown automata situation is different.
- We focus on nondeterministic PDA because these automata are equivalent in power to CFG.

Formal Definition of a Pushdown Automaton

Definition (PDA (下推自动机))

Formal Definition of a Pushdown Automaton

Definition (PDA (下推自动机))

A ***pushdown automaton*** (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

Formal Definition of a Pushdown Automaton

Definition (PDA (下推自动机))

A ***pushdown automaton*** (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of ***states***,

Formal Definition of a Pushdown Automaton

Definition (PDA (下推自动机))

A ***pushdown automaton*** (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of ***states***,
2. Σ is a finite set called the ***input alphabet***,

Formal Definition of a Pushdown Automaton

Definition (PDA (下推自动机))

A ***pushdown automaton*** (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of ***states***,
2. Σ is a finite set called the ***input alphabet***,
3. Γ is a finite set called the ***stack alphabet***,

Formal Definition of a Pushdown Automaton

Definition (PDA (下推自动机))

A ***pushdown automaton*** (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of ***states***,
2. Σ is a finite set called the ***input alphabet***,
3. Γ is a finite set called the ***stack alphabet***,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the ***transition function***,

Formal Definition of a Pushdown Automaton

Definition (PDA (下推自动机))

A ***pushdown automaton*** (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of ***states***,
2. Σ is a finite set called the ***input alphabet***,
3. Γ is a finite set called the ***stack alphabet***,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the ***transition function***,
5. $q_0 \in Q$ is the ***start state***, and

Formal Definition of a Pushdown Automaton

Definition (PDA (下推自动机))

A ***pushdown automaton*** (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

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

Formal Definition of Computation for a PDA

A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows.

- It accepts input w if w can be written as $w = w_1 w_2 \cdots w_m$, where $w_i \in \Sigma_\varepsilon$ and

Formal Definition of Computation for a PDA

A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows.

- It accepts input w if w can be written as $w = w_1 w_2 \cdots w_m$, where $w_i \in \Sigma_\varepsilon$ and
- sequences of states $r_0, r_1, \ldots, r_m \in Q$ and strings $s_0, s_1, \ldots, s_m \in \Gamma^*$ exist that satisfy the following three conditions.

Formal Definition of Computation for a PDA

A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows.

- It accepts input w if w can be written as $w = w_1 w_2 \cdots w_m$, where $w_i \in \Sigma_\varepsilon$ and
- sequences of states $r_0, r_1, \ldots, r_m \in Q$ and strings $s_0, s_1, \ldots, s_m \in \Gamma^*$ exist that satisfy the following three conditions.
- The strings s_i represent the sequence of stack contents that M has on the accepting branch of the computation.

Formal Definition of Computation for a PDA

A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows.

- It accepts input w if w can be written as $w = w_1 w_2 \cdots w_m$, where $w_i \in \Sigma_\varepsilon$ and
- sequences of states $r_0, r_1, \ldots, r_m \in Q$ and strings $s_0, s_1, \ldots, s_m \in \Gamma^*$ exist that satisfy the following three conditions.
- The strings s_i represent the sequence of stack contents that M has on the accepting branch of the computation.

1. $r_0 = q_0$ and $s_0 = \varepsilon$

Formal Definition of Computation for a PDA

A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows.

- It accepts input w if w can be written as $w = w_1 w_2 \cdots w_m$, where $w_i \in \Sigma_\varepsilon$ and
- sequences of states $r_0, r_1, \ldots, r_m \in Q$ and strings $s_0, s_1, \ldots, s_m \in \Gamma^*$ exist that satisfy the following three conditions.
- The strings s_i represent the sequence of stack contents that M has on the accepting branch of the computation.
  1. $r_0 = q_0$ and $s_0 = \varepsilon$
  2. For $i = 0, \ldots, m-1$, we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$

Formal Definition of Computation for a PDA

A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows.

- It accepts input w if w can be written as $w = w_1 w_2 \cdots w_m$, where $w_i \in \Sigma_\varepsilon$ and
- sequences of states $r_0, r_1, \ldots, r_m \in Q$ and strings $s_0, s_1, \ldots, s_m \in \Gamma^*$ exist that satisfy the following three conditions.
- The strings s_i represent the sequence of stack contents that M has on the accepting branch of the computation.

1. $r_0 = q_0$ and $s_0 = \varepsilon$
2. For $i = 0, \ldots, m-1$, we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$
3. $r_m \in F$

Examples of Pushdown Automata

Example (PDA M_1 recognizes the language $\{0^n1^n \mid n \geq 0\}$)

Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, \$\}$

$F = \{q_1, q_4\}$, and

δ is given by the following table, wherein blank entries signify $\emptyset$

Examples of Pushdown Automata

Example (PDA M_1 recognizes the language $\{0^n 1^n \mid n \geq 0\}$)

Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, \$\}$

$F = \{q_1, q_4\}$, and

δ is given by the following table, wherein blank entries signify $\emptyset$

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									

Examples of Pushdown Automata

Example (PDA M_1 recognizes the language $\{0^n1^n \mid n \geq 0\}$)

Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$

$\Sigma = \{0, 1\}$ $\Gamma = \{0, \$\}$ $F = \{q_1, q_4\}$, and

State diagram for the PDA M_1

Examples of Pushdown Automata

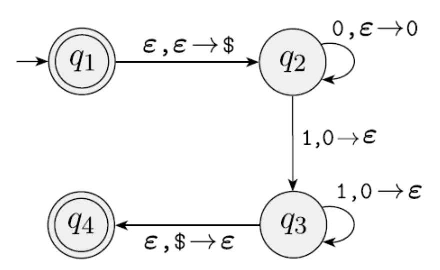
Example (PDA M_1 recognizes the language $\{0^n 1^n \mid n \geq 0\}$)

Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$

$\Sigma = \{0, 1\}$ $\Gamma = \{0, \$\}$ $F = \{q_1, q_4\}$, and

State diagram for the PDA M_1



Examples of Pushdown Automata

Example (PDA M_2)

A pushdown automaton that recognizes the language

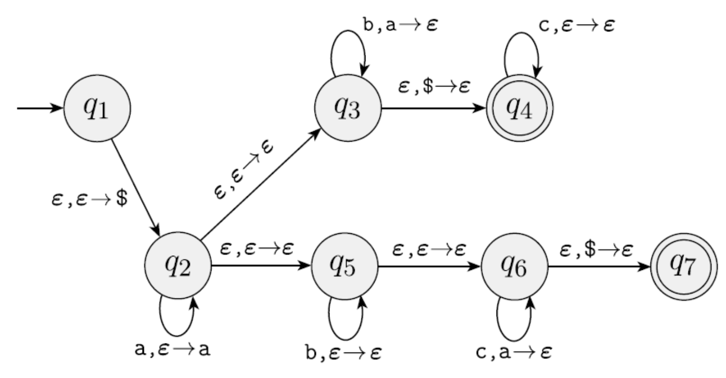
$\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

Examples of Pushdown Automata

Example (PDA M_2)

A pushdown automaton that recognizes the language

$\{a^i b^j c^k \mid i, j, k \geq 0$ and $i = j$ or $i = k\}$



Examples of Pushdown Automata

Example (PDA M_3)

A pushdown automaton that recognizes the language

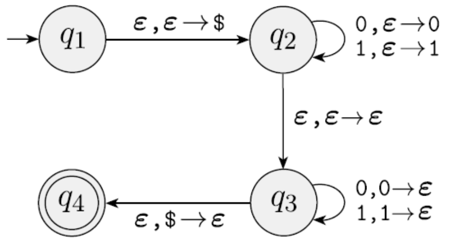
$\{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$

Examples of Pushdown Automata

Example (PDA M_3)

A pushdown automaton that recognizes the language

$\{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$



Equivalence With Context-Free Grammars

Theorem

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

Equivalence With Context-Free Grammars

Theorem

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

Lemma

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

Equivalence With Context-Free Grammars

Theorem

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

Lemma

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

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- Let A be a CFL generated by a CFG G. We convert G into an equivalent PDA P.

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- Let A be a CFL generated by a CFG G. We convert G into an equivalent PDA P.
- P accepts a input w, if G generates w by a sequence of derivations.

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- Let A be a CFL generated by a CFG G. We convert G into an equivalent PDA P.
- P accepts a input w, if G generates w by a sequence of derivations.
- PDA P begins by writing the start variable on its stack. It goes through a series of intermediate strings. Eventually it may arrive at a string that contains only terminal symbols. Then P accepts if this string is identical to the string it has received as input.

Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

The following is an informal description of P.

Equivalence With Context-Free Grammars

Lemma

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

The following is an informal description of P.

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

Equivalence With Context-Free Grammars

Lemma

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

The following is an informal description of P.

1. Place the marker symbol \$ and the start variable on the stack.
2. Repeat the following steps forever.

Equivalence With Context-Free Grammars

Lemma

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

The following is an informal description of P.

1. Place the marker symbol $\$$ and the start variable on the stack.
2. Repeat the following steps forever.
    1. 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.

Equivalence With Context-Free Grammars

Lemma

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

The following is an informal description of P.

1. Place the marker symbol $\$$ and the start variable on the stack.
2. Repeat the following steps forever.
   1. 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.
   2. 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.

Equivalence With Context-Free Grammars

Lemma

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

The following is an informal description of P.

1. Place the marker symbol $\$$ and the start variable on the stack.
2. Repeat the following steps forever.
   1. 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.
   2. 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.
   3. If the top of stack is the symbol $\$$, enter the accept state. Doing so accepts the input if it has all been read.

Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- Construct a pushdown automation $P = (Q, \Sigma, \Gamma, q_{\text{start}}, F)$. We use a shorthand that provides a way to write an entire string on the stack in one step of the machine.

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- Construct a pushdown automation $P = (Q, \Sigma, \Gamma, q_{\text{start}}, F)$. We use a shorthand that provides a way to write an entire string on the stack in one step of the machine.
- Let q and r be states of the PDA and let a be in Σ_ϵ and s be in Γ_ϵ. P goes from q to r when it reads a and pops s.

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- Construct a pushdown automation $P = (Q, \Sigma, \Gamma, q_{\text{start}}, F)$. We use a shorthand that provides a way to write an entire string on the stack in one step of the machine.
- Let q and r be states of the PDA and let a be in Σ_ϵ and s be in Γ_ϵ. P goes from q to r when it reads a and pops s.
- Push the entire string $u = u_1, \cdots, u_l$ on the stack at the same time.

□

Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- Implement this action by introducing new states $q_1, \cdots, q_{l-1}$ and setting the transition function as follows:

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- Implement this action by introducing new states $q_1, \cdots, q_{l-1}$ and setting the transition function as follows:

$\delta(q, a, s)$ to contain (q_1, u_l),

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- Implement this action by introducing new states $q_1, \cdots, q_{l-1}$ and setting the transition function as follows:

$\delta(q, a, s)$ to contain (q_1, u_l),

$$\delta(q_1, \epsilon, \epsilon) = \{(q_2, u_{l-1})\},$$

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- Implement this action by introducing new states $q_1, \cdots, q_{l-1}$ and setting the transition function as follows:

$\delta(q, a, s)$ to contain (q_1, u_l),

$$\delta(q_1, \epsilon, \epsilon) = \{(q_2, u_{l-1})\},$$

$$\delta(q_2, \epsilon, \epsilon) = \{(q_3, u_{l-2})\},$$

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- Implement this action by introducing new states $q_1, \cdots, q_{l-1}$ and setting the transition function as follows:

$\delta(q, a, s)$ to contain (q_1, u_l),

$$\delta(q_1, \epsilon, \epsilon) = \{(q_2, u_{l-1})\},$$
$$\delta(q_2, \epsilon, \epsilon) = \{(q_3, u_{l-2})\},$$
$$\cdots,$$
$$\delta(q_{l-1}, \epsilon, \epsilon) = \{(r, u_1)\},$$

Equivalence With Context-Free Grammars

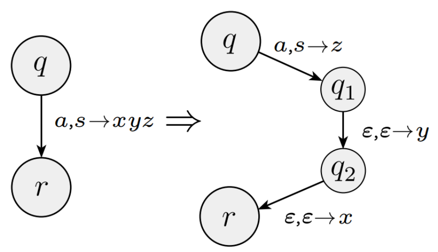
Lemma

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

Proof.

- Implement this action by introducing new states $q_1, \cdots, q_{l-1}$ and setting the transition function as follows:

$$\delta(q, a, s) \text{ to contain } (q_1, u_l),$$
$$\delta(q_1, \epsilon, \epsilon) = \{(q_2, u_{l-1})\},$$
$$\delta(q_2, \epsilon, \epsilon) = \{(q_3, u_{l-2})\},$$
$$\cdots,$$
$$\delta(q_{l-1}, \epsilon, \epsilon) = \{(r, u_1)\},$$



□

Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- We use the notation $(r, u) \in \delta(q, a, s)$ to mean that when q is the state of the automaton, a is the next input symbol, and s is the symbol on the top of the stack, the PDA may read the a and pop the s, then push the string u onto the stack and go on to the state r.

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- We use the notation $(r, u) \in \delta(q, a, s)$ to mean that when q is the state of the automaton, a is the next input symbol, and s is the symbol on the top of the stack, the PDA may read the a and pop the s, then push the string u onto the stack and go on to the state r.
- The states of P are $Q = \{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}\} \cup E$, E is s the set of states we need for implementing the shorthand just described.

$\square$

Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- The stack is initialized to contain $\$$ and S, implementing step 1 in the informal description:

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- The stack is initialized to contain $\$$ and S, implementing step 1 in the informal description:

  $\delta(q_{\text{start}}, \epsilon, \epsilon) = \{(q_{\text{loop}}, S\$)\}$.

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- The stack is initialized to contain $\$$ and S, implementing step 1 in the informal description:

  $\delta(q_{\text{start}}, \epsilon, \epsilon) = \{(q_{\text{loop}}, S\$)\}$.
- Main loop of step 2:

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- The stack is initialized to contain $\$$ and S, implementing step 1 in the informal description:

  $\delta(q_{\text{start}}, \epsilon, \epsilon) = \{(q_{\text{loop}}, S\$)\}$.
- Main loop of step 2:
  - $\delta(q_{\text{loop}}, \epsilon, A) = \{(q_{\text{loop},w}) | \text{where } A \to w \text{ is a rule in } R\}$, the top of the stack contains a variable.

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- The stack is initialized to contain $\$$ and S, implementing step 1 in the informal description:
  $\delta(q_{\text{start}}, \epsilon, \epsilon) = \{(q_{\text{loop}}, S\$)\}$.
- Main loop of step 2:
  - $\delta(q_{\text{loop}}, \epsilon, A) = \{(q_{\text{loop},w}) | \text{where } A \rightarrow w \text{ is a rule in } R\}$, the top of the stack contains a variable.
  - $\delta(q_{\text{loop}}, a, a) = \{(q_{\text{loop}}, \epsilon)\}$, the top of the stack contains a terminal.

Equivalence With Context-Free Grammars

Lemma

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

Proof.

- The stack is initialized to contain $\$$ and S, implementing step 1 in the informal description:

  $\delta(q_{\text{start}}, \epsilon, \epsilon) = \{(q_{\text{loop}}, S\$)\}$.
- Main loop of step 2:
  - $\delta(q_{\text{loop}}, \epsilon, A) = \{(q_{\text{loop},w}) |$where $A \rightarrow w$ is a rule in $R\}$, the top of the stack contains a variable.
  - $\delta(q_{\text{loop}}, a, a) = \{(q_{\text{loop}}, \epsilon)\}$, the top of the stack contains a terminal.
  - $\delta(q_{\text{loop}}, \epsilon, \$) = \{(q_{\text{accept}}, \epsilon)\}$, the empty stack marker $\$$ is on the top of the stack.

Equivalence With Context-Free Grammars

Lemma

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

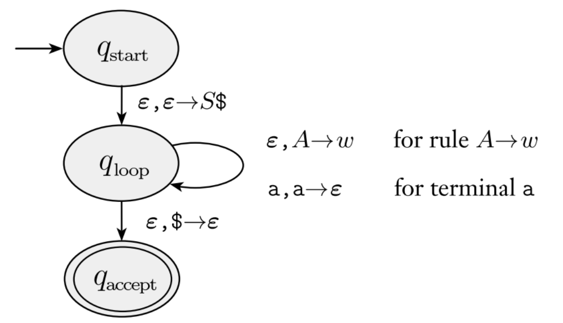
Equivalence With Context-Free Grammars

Lemma

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

Proof.

The state diagram is shown in the following figure



Equivalence With Context-Free Grammars

Example (construct a PDA P from the following CFG G.)

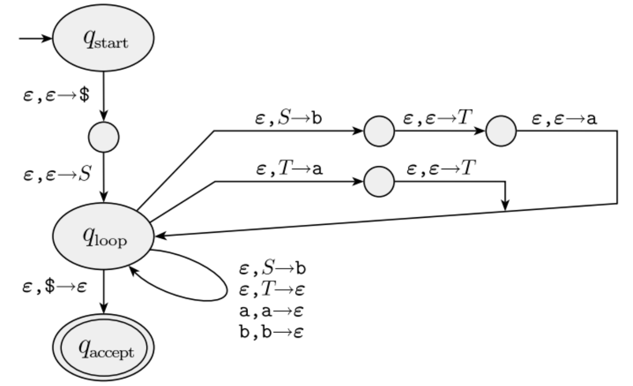
$S \rightarrow aTb|b$, $T \rightarrow Ta|\epsilon$

Equivalence With Context-Free Grammars

Example (construct a PDA P from the following CFG G.)

$S \to aTb|b$, $T \to Ta|\epsilon$

The transition function is shown in the following diagram



Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- Given a PDA P, make a CFG G generating all the strings that P accepts.

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- Given a PDA P, make a CFG G generating all the strings that P accepts.
- P accepts a input w, if G generates w by a sequence of derivations.

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- Given a PDA P, make a CFG G generating all the strings that P accepts.
- P accepts a input w, if G generates w by a sequence of derivations.
- Design a grammar that does somewhat more. For each pair of states p and q in P, the grammar will have a variable A_{pq}. This variable generates all the strings that can take P from p with an empty stack to q with an empty stack.

Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- We simplify our task by modifying P slightly to give it the following three features:

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- We simplify our task by modifying P slightly to give it the following three features:
  - It has a single accept state, q_{accept}.

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- We simplify our task by modifying P slightly to give it the following three features:
  - It has a single accept state, q_{accept}.
  - It empties its stack before accepting.

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- We simplify our task by modifying P slightly to give it the following three features:
  - It has a single accept state, q_{accept}.
  - It empties its stack before accepting.
  - Each transition either pushes a symbol onto the stack (a push move) or pops one off the stack (a pop move), but it does not do both at the same time.

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- We simplify our task by modifying P slightly to give it the following three features:
  - It has a single accept state, q_{accept}.
  - It empties its stack before accepting.
  - Each transition either pushes a symbol onto the stack (a push move) or pops one off the stack (a pop move), but it does not do both at the same time.

Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- Giving P features 1 and 2 is easy.

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- Giving P features 1 and 2 is easy.
- To give it feature 3,

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- Giving P features 1 and 2 is easy.
- To give it feature 3,
  - we replace each transition that simultaneously pops and pushes with a two transition sequence that goes through a new state;

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- Giving P features 1 and 2 is easy.
- To give it feature 3,
  - we replace each transition that simultaneously pops and pushes with a two transition sequence that goes through a new state;
  - we replace each transition that neither pops nor pushes with a two transition sequence that pushes then pops an arbitrary stack symbol.

Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- Two possibilities occur during P's computation on x. Either the symbol popped at the end is the symbol that was pushed at the beginning, or not.

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- Two possibilities occur during P's computation on x. Either the symbol popped at the end is the symbol that was pushed at the beginning, or not.
  - Simulate the former possibility with the rule $A_{pq} \to aA_{rs}b$;

Equivalence With Context-Free Grammars

Lemma

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

Proof idea:

- Two possibilities occur during P's computation on x. Either the symbol popped at the end is the symbol that was pushed at the beginning, or not.
  - Simulate the former possibility with the rule $A_{pq} \to aA_{rs}b$;
  - We simulate the latter possibility with the rule $A_{pq} \to A_{pr}A_{rq}$.

Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Proof.

Given $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$, construct G. The variables of G are $\{A_{pq} | p, q \in Q\}$. The start variable is $A_{q_0, q_{\text{accept}}}$. We describe G's rules in three parts.

Equivalence With Context-Free Grammars

Lemma

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

Proof.

Given $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$, construct G. The variables of G are $\{A_{pq} | p, q \in Q\}$. The start variable is $A_{q_0, q_{\text{accept}}}$. We describe G's rules in three parts.

- For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma_\epsilon$, if $\delta p, a, \epsilon$ contains (r, u) and $\delta(s, b, u)$ contains (q, ϵ) put the rule $A_{pq} \rightarrow aA_{rs}b$ in G.

Equivalence With Context-Free Grammars

Lemma

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

Proof.

Given $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$, construct G. The variables of G are $\{A_{pq} | p, q \in Q\}$. The start variable is $A_{q_0, q_{\text{accept}}}$. We describe G's rules in three parts.

- For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma_\epsilon$, if $\delta p, a, \epsilon$ contains (r, u) and $\delta(s, b, u)$ contains (q, ϵ) put the rule $A_{pq} \to aA_{rs}b$ in G.
- For each $p, q, r \in Q$, put the rule $A_{pq} \to A_{pr}A_{rq}$ in G.

Equivalence With Context-Free Grammars

Lemma

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

Proof.

Given $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$, construct G. The variables of G are $\{A_{pq} | p, q \in Q\}$. The start variable is $A_{q_0, q_{\text{accept}}}$. We describe G's rules in three parts.

- For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma_\epsilon$, if $\delta p, a, \epsilon$ contains (r, u) and $\delta(s, b, u)$ contains (q, ϵ) put the rule $A_{pq} \to aA_{rs}b$ in G.
- For each $p, q, r \in Q$, put the rule $A_{pq} \to A_{pr}A_{rq}$ in G.
- Finally, for each $p \in Q$, put the rule $A_{pp} \to \epsilon$ in G.

$\square$

Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Proof.

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

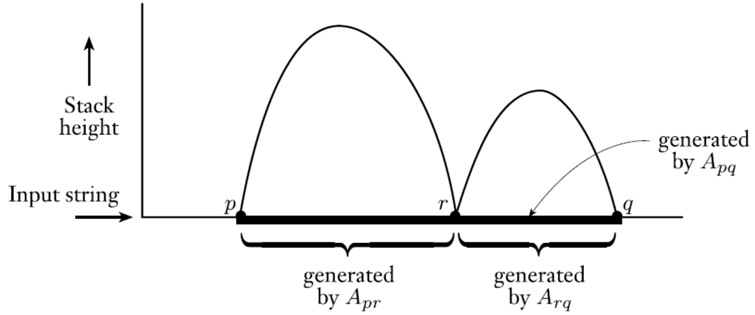
Equivalence With Context-Free Grammars

Lemma

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

Proof.

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



Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Proof.

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

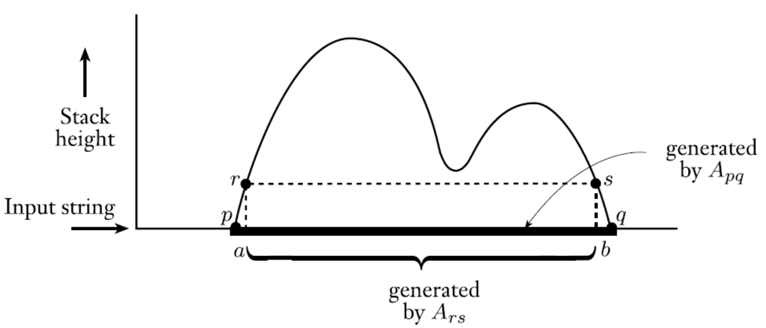
Equivalence With Context-Free Grammars

Lemma

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

Proof.

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



Equivalence With Context-Free Grammars

Lemma

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

Equivalence With Context-Free Grammars

Lemma

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

Claim

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

Equivalence With Context-Free Grammars

Lemma

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

Claim

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

Claim

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

Equivalence With Context-Free Grammars

Claim

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

Equivalence With Context-Free Grammars

Claim

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

Proof.

We prove this claim by induction on the number of steps in the derivation of x from A_{pq}.

Equivalence With Context-Free Grammars

Claim

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

Proof.

We prove this claim by induction on the number of steps in the derivation of x from A_{pq}.

- ***Basis***: The derivation has 1 step. A derivation with a single step must use a rule whose right-hand side contains no variables. The only rules in G where no variables occur on the right-hand side are $A_{pp} \to \epsilon$.

□

Equivalence With Context-Free Grammars

Proof.

Induction step: Assume true for derivations of length at most k, where $k \geq 1$, and prove true for derivations of length $k+1$. Suppose $A_{pq} \overset{*}{\Rightarrow} x$ with $k+1$ steps. The first step in this derivation is either $A_{pq} \Rightarrow aA_{rs}b$ or $A_{pq} \Rightarrow A_{pr}A_{rq}$. We handle these two cases separately.

Equivalence With Context-Free Grammars

Proof.

Induction step: Assume true for derivations of length at most k, where $k \geq 1$, and prove true for derivations of length $k+1$. Suppose $A_{pq} \overset{*}{\Rightarrow} x$ with $k+1$ steps. The first step in this derivation is either $A_{pq} \Rightarrow aA_{rs}b$ or $A_{pq} \Rightarrow A_{pr}A_{rq}$. We handle these two cases separately.

In the first case, consider the portion y of x that A_{rs} generates, $x = ayb$.

Equivalence With Context-Free Grammars

Proof.

Induction step: Assume true for derivations of length at most k, where $k \geq 1$, and prove true for derivations of length $k+1$. Suppose $A_{pq} \overset{*}{\Rightarrow} x$ with $k+1$ steps. The first step in this derivation is either $A_{pq} \Rightarrow aA_{rs}b$ or $A_{pq} \Rightarrow A_{pr}A_{rq}$. We handle these two cases separately.

In the first case, consider the portion y of x that A_{rs} generates, $x = ayb$.

- $A_{rs} \overset{*}{\Rightarrow} y$ with k steps, then P can go from r on empty stack to s on empty stack.

Equivalence With Context-Free Grammars

Proof.

Induction step: Assume true for derivations of length at most k, where $k \geq 1$, and prove true for derivations of length $k+1$. Suppose $A_{pq} \overset{*}{\Rightarrow} x$ with $k+1$ steps. The first step in this derivation is either $A_{pq} \Rightarrow aA_{rs}b$ or $A_{pq} \Rightarrow A_{pr}A_{rq}$. We handle these two cases separately.

In the first case, consider the portion y of x that A_{rs} generates, $x = ayb$.

- $A_{rs} \overset{*}{\Rightarrow} y$ with k steps, then P can go from r on empty stack to s on empty stack.
- Because $A_{pq} \to aA_{rs}b$ is a rule in G, $\delta(p, a, \epsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ϵ), for some stack symbol u.

Equivalence With Context-Free Grammars

Proof.

Induction step: Assume true for derivations of length at most k, where $k \geq 1$, and prove true for derivations of length $k+1$. Suppose $A_{pq} \overset{*}{\Rightarrow} x$ with $k+1$ steps. The first step in this derivation is either $A_{pq} \Rightarrow aA_{rs}b$ or $A_{pq} \Rightarrow A_{pr}A_{rq}$. We handle these two cases separately.

In the first case, consider the portion y of x that A_{rs} generates, $x = ayb$.

- $A_{rs} \overset{*}{\Rightarrow} y$ with k steps, then P can go from r on empty stack to s on empty stack.
- Because $A_{pq} \to aA_{rs}b$ is a rule in G, $\delta(p, a, \epsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ϵ), for some stack symbol u.
- x can bring it from p with empty stack to q with empty stack.

□

Equivalence With Context-Free Grammars

Claim

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

Proof.

In the second case, consider the portions y and z of x that A_{pr} and A_{rq} respectively generate, $x = yz$.

- $A_{pr} \overset{*}{\Rightarrow} y$ in at most k steps and $A_{rq} \overset{*}{\Rightarrow} z$ in at most k steps.

Equivalence With Context-Free Grammars

Claim

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

Proof.

In the second case, consider the portions y and z of x that A_{pr} and A_{rq} respectively generate, $x = yz$.

- $A_{pr} \overset{*}{\Rightarrow} y$ in at most k steps and $A_{rq} \overset{*}{\Rightarrow} z$ in at most k steps.
- y can bring P from p to r, and z can bring P from r to q, with empty stacks at the beginning and end.

Equivalence With Context-Free Grammars

Claim

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

Proof.

In the second case, consider the portions y and z of x that A_{pr} and A_{rq} respectively generate, $x = yz$.

- $A_{pr} \stackrel{*}{\Rightarrow} y$ in at most k steps and $A_{rq} \stackrel{*}{\Rightarrow} z$ in at most k steps.
- y can bring P from p to r, and z can bring P from r to q, with empty stacks at the beginning and end.
- x can bring it from p with empty stack to q with empty stack.

Equivalence With Context-Free Grammars

Claim

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

Proof.

In the second case, consider the portions y and z of x that A_{pr} and A_{rq} respectively generate, $x = yz$.

- $A_{pr} \overset{*}{\Rightarrow} y$ in at most k steps and $A_{rq} \overset{*}{\Rightarrow} z$ in at most k steps.
- y can bring P from p to r, and z can bring P from r to q, with empty stacks at the beginning and end.
- x can bring it from p with empty stack to q with empty stack.

This completes the induction step. □

Equivalence With Context-Free Grammars

Claim

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

Equivalence With Context-Free Grammars

Claim

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

Proof.

We prove this claim by induction on the number of steps in the computation of P that goes from p to q with empty stacks on input x.

Equivalence With Context-Free Grammars

Claim

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

Proof.

We prove this claim by induction on the number of steps in the computation of P that goes from p to q with empty stacks on input x.

Basis: The computation has 0 steps.

Equivalence With Context-Free Grammars

Claim

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

Proof.

We prove this claim by induction on the number of steps in the computation of P that goes from p to q with empty stacks on input x.

Basis: The computation has 0 steps.

If a computation has 0 steps, it starts and ends at the same p. We show that $A_{pp} \overset{*}{\Rightarrow} x$. In 0 steps, P cannot read any characters, so $x = \epsilon$. By construction, G has the rule $A_{pp} \rightarrow \epsilon$, so the basis is proved. □

Equivalence With Context-Free Grammars

Claim

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

Proof.

Induction step: Assume true for computations of length at most k, where $k \geq 0$, and prove true for computations of length $k+1$.

Equivalence With Context-Free Grammars

Claim

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

Proof.

Induction step: Assume true for computations of length at most k, where $k \geq 0$, and prove true for computations of length $k+1$.

Suppose that P has a computation wherein x brings p to q with empty stacks in $k+1$ steps. Either the stack is empty only at the beginning and end of this computation, or it becomes empty elsewhere, too.

□

Equivalence With Context-Free Grammars

Proof.

In the first case,

Equivalence With Context-Free Grammars

Proof.

In the first case,

- The symbol that is pushed at the first move must be the same as the symbol that is popped at the last move, called it u.

Equivalence With Context-Free Grammars

Proof.

In the first case,

- The symbol that is pushed at the first move must be the same as the symbol that is popped at the last move, called it u.
- $\delta(p, a, \epsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ϵ), and so rule $A_{pq} \to aA_{rs}b$ is in G.

Equivalence With Context-Free Grammars

Proof.

In the first case,

- The symbol that is pushed at the first move must be the same as the symbol that is popped at the last move, called it u.
- $\delta(p, a, \epsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ϵ), and so rule $A_{pq} \rightarrow aA_{rs}b$ is in G.
- Consider $x = ayb$, P can go from r with an empty stack to s with an empty stack on input y, the computation on y has $(k+1) - 2 = k - 1$ steps.

Equivalence With Context-Free Grammars

Proof.

In the first case,

- The symbol that is pushed at the first move must be the same as the symbol that is popped at the last move, called it u.
- $\delta(p, a, \epsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ϵ), and so rule $A_{pq} \to aA_{rs}b$ is in G.
- Consider $x = ayb$, P can go from r with an empty stack to s with an empty stack on input y, the computation on y has $(k+1) - 2 = k - 1$ steps.
- $A_{rs} \overset{*}{\Rightarrow} y$ and hence $A_{pq} \overset{*}{\Rightarrow} x$

□

Equivalence With Context-Free Grammars

Proof.

In the second case,

Equivalence With Context-Free Grammars

Proof.

In the second case,

- let r be a state where the stack becomes empty other than at the beginning or end of the computation on x.

Equivalence With Context-Free Grammars

Proof.

In the second case,

- let r be a state where the stack becomes empty other than at the beginning or end of the computation on x.
- The portions of the computation from p to r and from r to q each contain at most k steps.

Equivalence With Context-Free Grammars

Proof.

In the second case,

- let r be a state where the stack becomes empty other than at the beginning or end of the computation on x.
- The portions of the computation from p to r and from r to q each contain at most k steps.
- y is the input read during the first portion and z is the input read during the second portion.

Equivalence With Context-Free Grammars

Proof.

In the second case,

- let r be a state where the stack becomes empty other than at the beginning or end of the computation on x.
- The portions of the computation from p to r and from r to q each contain at most k steps.
- y is the input read during the first portion and z is the input read during the second portion.
- We have $A_{pr} \overset{*}{\Rightarrow} y$ and $A_{rq} \overset{*}{\Rightarrow} z$. Because $A_{pq} \rightarrow A_{pr}A_{rq}$ in G, then $A_{pq} \overset{*}{\Rightarrow} x$.

Equivalence With Context-Free Grammars

Proof.

In the second case,

- let r be a state where the stack becomes empty other than at the beginning or end of the computation on x.
- The portions of the computation from p to r and from r to q each contain at most k steps.
- y is the input read during the first portion and z is the input read during the second portion.
- We have $A_{pr} \overset{*}{\Rightarrow} y$ and $A_{rq} \overset{*}{\Rightarrow} z$. Because $A_{pq} \rightarrow A_{pr}A_{rq}$ in G, then $A_{pq} \overset{*}{\Rightarrow} x$.

Proof is complete. □

Equivalence With Context-Free Grammars

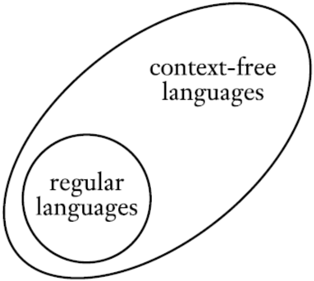
Corollary

Every regular language is context free.

Equivalence With Context-Free Grammars

Corollary

Every regular language is context free.



Outline

1. Context-Free Grammars
2. Pushdown Automata
3. Non-Context-Free Languages
   - The Pumping Lemma for Context-Free Languages
4. Deterministic Context-Free Languages

The Pumping Lemma for Context-Free Languages

Theorem (Pumping lemma for context-free languages 泵引理)

If A is a context-free language, then 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 conditions:

The Pumping Lemma for Context-Free Languages

Theorem (Pumping lemma for context-free languages 泵引理)

If A is a context-free language, then 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 conditions:

1. *for each $i \geq 0$, $uv^ixy^iz \in A$,*

The Pumping Lemma for Context-Free Languages

Theorem (Pumping lemma for context-free languages 泵引理)

If A is a context-free language, then 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 conditions:

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

The Pumping Lemma for Context-Free Languages

Theorem (Pumping lemma for context-free languages 泵引理)

If A is a context-free language, then 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 conditions:

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

The Pumping Lemma for Context-Free Languages

Proof idea: Let A be a CFL and let G be a CFG that generates it. We must show that any sufficiently long string s in A can be pumped and remain in A.

The Pumping Lemma for Context-Free Languages

Proof idea: Let A be a CFL and let G be a CFG that generates it. We must 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, it is derivable from G and has a parse tree. The parse tree must contain a long path from the root to one of a leaf.

The Pumping Lemma for Context-Free Languages

Proof idea: Let A be a CFL and let G be a CFG that generates it. We must 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, it is derivable from G and has a parse tree. The parse tree must contain a long path from the root to one of a leaf.
- On this long path, some variable symbol R must repeat because of the pigeonhole principle.

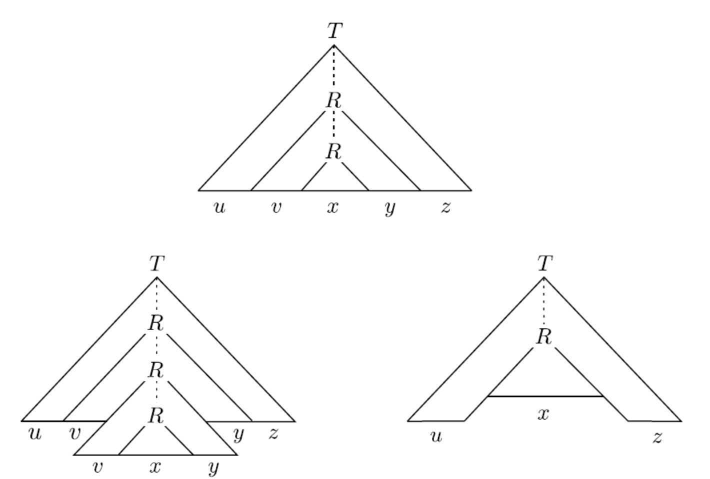
The Pumping Lemma for Context-Free Languages

Proof idea: Let A be a CFL and let G be a CFG that generates it. We must 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, it is derivable from G and has a parse tree. The parse tree must contain a long path from the root to one of a leaf.
- On this long path, some variable symbol R must repeat because of the pigeonhole principle.
- Replace the subtree under the second occurrence of R with the subtree under the first occurrence of R and still get a legal parse tree.

The Pumping Lemma for Context-Free Languages

Example (Surgery on parse trees)



The Pumping Lemma for Context-Free Languages

Proof.

Let G be a CFG for CFL A. Let b be the maximum number of symbols in the right-hand side of a rule.

- A node can have no more than b children. It means at most b^h leaves are within h steps of the start variable (the root of the parse tree).

The Pumping Lemma for Context-Free Languages

Proof.

Let G be a CFG for CFL A. Let b be the maximum number of symbols in the right-hand side of a rule.

- A node can have no more than b children. It means at most b^h leaves are within h steps of the start variable (the root of the parse tree).
- If the height of the parse tree is at most h, the length of the string generated is at most b^h.

The Pumping Lemma for Context-Free Languages

Proof.

Let G be a CFG for CFL A. Let b be the maximum number of symbols in the right-hand side of a rule.

- A node can have no more than b children. It means at most b^h leaves are within h steps of the start variable (the root of the parse tree).
- If the height of the parse tree is at most h, the length of the string generated is at most b^h.
- Let V denote the number of variables in G, we set p, the pumping length, to be $b^{|V|+1}$.

□

The Pumping Lemma for Context-Free Languages

Proof.

To see how to pump any such string s

The Pumping Lemma for Context-Free Languages

Proof.

To see how to pump any such string s

- let τ be one of its parse trees. If s has several parse trees, choose τ to be a parse tree that has the smallest number of nodes.

The Pumping Lemma for Context-Free Languages

Proof.

To see how to pump any such string s

- let τ be one of its parse trees. If s has several parse trees, choose τ to be a parse tree that has the smallest number of nodes.
- τ must be at least $|V| + 1$ high, so its longest path from the root to a leaf has length at least $|V| + 1$.

The Pumping Lemma for Context-Free Languages

Proof.

To see how to pump any such string s

- let τ be one of its parse trees. If s has several parse trees, choose τ to be a parse tree that has the smallest number of nodes.
- τ must be at least $|V|+1$ high, so its longest path from the root to a leaf has length at least $|V|+1$.
- The path has at least $|V|+2$ nodes and hence this path has at least $|V|+1$ variables.

□

The Pumping Lemma for Context-Free Languages

Proof.

- With G having only $|V|$ variables, some variable R appears more than once on that path.

The Pumping Lemma for Context-Free Languages

Proof.

- With G having only $|V|$ variables, some variable R appears more than once on that path.
- we select R to be a variable that repeats among the lowest $|V|+1$ variables on this path.

The Pumping Lemma for Context-Free Languages

Proof.

- With G having only $|V|$ variables, some variable R appears more than once on that path.
- we select R to be a variable that repeats among the lowest $|V| + 1$ variables on this path.
- The path has at least $|V| + 2$ nodes and hence this path has at least $|V| + 1$ variables.

$\square$

The Pumping Lemma for Context-Free Languages

Proof.

We divide s into $uvxyz$ according the Figure.

The Pumping Lemma for Context-Free Languages

Proof.

We divide s into $uvxyz$ according the Figure.

- Each occurrence of R has a subtree under it, generating a part of the string s.

The Pumping Lemma for Context-Free Languages

Proof.

We divide s into $uvxyz$ according the Figure.

- Each occurrence of R has a subtree under it, generating a part of the string s.
- The upper occurrence of R has a larger subtree and generates vxy, whereas the lower occurrence generates just x with a smaller subtree.

The Pumping Lemma for Context-Free Languages

Proof.

We divide s into $uvxyz$ according the Figure.

- Each occurrence of R has a subtree under it, generating a part of the string s.
- The upper occurrence of R has a larger subtree and generates vxy, whereas the lower occurrence generates just x with a smaller subtree.
- We substitute one for the other and still obtain a valid parse tree.

The Pumping Lemma for Context-Free Languages

Proof.

We divide s into $uvxyz$ according the Figure.

- Each occurrence of R has a subtree under it, generating a part of the string s.
- The upper occurrence of R has a larger subtree and generates vxy, whereas the lower occurrence generates just x with a smaller subtree.
- We substitute one for the other and still obtain a valid parse tree.

That establishes condition 1 of the lemma $\square$

The Pumping Lemma for Context-Free Languages

Proof.

For condition 2, we must be sure that v and y are not both ϵ.

The Pumping Lemma for Context-Free Languages

Proof.

For condition 2, we must be sure that v and y are not both ϵ.

- If they were, the parse tree obtained by substituting the smaller subtree for the larger would have fewer nodes than τ does and would still generate s.

The Pumping Lemma for Context-Free Languages

Proof.

For condition 2, we must be sure that v and y are not both ϵ.

- If they were, the parse tree obtained by substituting the smaller subtree for the larger would have fewer nodes than τ does and would still generate s.
- This result isn't possible because we had already chosen τ to be a parse tree for s with the smallest number of nodes.

$\square$

The Pumping Lemma for Context-Free Languages

Proof.

For condition 3, we need to be sure that vxy has length at most p.

The Pumping Lemma for Context-Free Languages

Proof.

For condition 3, we need to be sure that vxy has length at most p.

- In the parse tree for s the upper occurrence of R generates vxy.

The Pumping Lemma for Context-Free Languages

Proof.

For condition 3, we need to be sure that vxy has length at most p.

- In the parse tree for s the upper occurrence of R generates vxy.
- We chose R such that both occurrences fall within the bottom $|V|+1$ variables on the path and chose the longest path in the parse tree.

The Pumping Lemma for Context-Free Languages

Proof.

For condition 3, we need to be sure that vxy has length at most p.

- In the parse tree for s the upper occurrence of R generates vxy.
- We chose R such that both occurrences fall within the bottom $|V| + 1$ variables on the path and chose the longest path in the parse tree.
- The subtree where R generates vxy is at most $|V| + 1$ high.

The Pumping Lemma for Context-Free Languages

Proof.

For condition 3, we need to be sure that vxy has length at most p.

- In the parse tree for s the upper occurrence of R generates vxy.
- We chose R such that both occurrences fall within the bottom $|V|+1$ variables on the path and chose the longest path in the parse tree.
- The subtree where R generates vxy is at most $|V|+1$ high.
- A tree of this height can generate a string of length at most $b^{|V|+1} = p$.

$\square$

The Pumping Lemma for Context-Free Languages

Example

Use the pumping lemma to show that the language $B = \{a^n b^n c^n | n \geq 0\}$ is not context free.

The Pumping Lemma for Context-Free Languages

Example

Use the pumping lemma to show that the language $B = \{a^n b^n c^n | n \geq 0\}$ is not context free.

Example

Let $C = \{a^i b^j c^k | 0 \leq i \leq j \leq k\}$. We use the pumping lemma to show that C is not a CFL.

The Pumping Lemma for Context-Free Languages

Example

Use the pumping lemma to show that the language $B = \{a^n b^n c^n | n \geq 0\}$ is not context free.

Example

Let $C = \{a^i b^j c^k | 0 \leq i \leq j \leq k\}$. We use the pumping lemma to show that C is not a CFL.

Example

Let $D = \{ww | w \in \{0,1\}^*\}$. Use the pumping lemma to show that D is not a CFL.

Outline

Deterministic Context-Free Languages

Deterministic Context-Free Languages

- The languages that are recognizable by deterministic pushdown automata (DPDAs) are called deterministic context-free languages (DCFLs).

Deterministic Context-Free Languages

- The languages that are recognizable by deterministic pushdown automata (DPDAs) are called deterministic context-free languages (DCFLs).
- Basic principle of determinism: at each step of its computation, the DPDA has at most one way to proceed according to its transition function.

Deterministic Context-Free Languages

- The languages that are recognizable by deterministic pushdown automata (DPDAs) are called deterministic context-free languages (DCFLs).
- Basic principle of determinism: at each step of its computation, the DPDA has at most one way to proceed according to its transition function.
- ϵ-moves is allowed in the DPDA' s transition function.

Deterministic Context-Free Languages

- The languages that are recognizable by deterministic pushdown automata (DPDAs) are called deterministic context-free languages (DCFLs).
- Basic principle of determinism: at each step of its computation, the DPDA has at most one way to proceed according to its transition function.
- ϵ-moves is allowed in the DPDA's transition function.
  - ϵ-input moves corresponding to $\delta(q, \epsilon, x)$;

Deterministic Context-Free Languages

- The languages that are recognizable by deterministic pushdown automata (DPDAs) are called deterministic context-free languages (DCFLs).
- Basic principle of determinism: at each step of its computation, the DPDA has at most one way to proceed according to its transition function.
- ϵ-moves is allowed in the DPDA’ s transition function.
    - ϵ-input moves corresponding to $\delta(q, \epsilon, x)$;
    - ϵ-stack moves corresponding to $\delta(q, a, \epsilon)$;

Deterministic Context-Free Languages

- The languages that are recognizable by deterministic pushdown automata (DPDAs) are called deterministic context-free languages (DCFLs).
- Basic principle of determinism: at each step of its computation, the DPDA has at most one way to proceed according to its transition function.
- ϵ-moves is allowed in the DPDA' s transition function.
  - ϵ-input moves corresponding to $\delta(q, \epsilon, x)$;
  - ϵ-stack moves corresponding to $\delta(q, a, \epsilon)$;
  - If a DPDA can make an ϵ-move in a certain situation, it is prohibited from making a move in that same situation that involves processing a symbol instead of ϵ.

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of ***states***,

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of ***states***,
2. Σ is a finite set called the ***input alphabet***,

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of ***states***,
2. Σ is a finite set called the ***input alphabet***,
3. Γ is a finite set called the ***stack alphabet***,

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of ***states***,
2. Σ is a finite set called the ***input alphabet***,
3. Γ is a finite set called the ***stack alphabet***,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to (Q \times \Gamma_\epsilon) \cup \{\emptyset\}$ is the ***transition function***,

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of ***states***,
2. Σ is a finite set called the ***input alphabet***,
3. Γ is a finite set called the ***stack alphabet***,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to (Q \times \Gamma_\epsilon) \cup \{\emptyset\}$ is the ***transition function***,
5. $q_0 \in Q$ is the ***start state***, and

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of ***states***,
2. Σ is a finite set called the ***input alphabet***,
3. Γ is a finite set called the ***stack alphabet***,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to (Q \times \Gamma_\epsilon) \cup \{\emptyset\}$ is the ***transition function***,
5. $q_0 \in Q$ is the ***start state***, and
6. $F \subseteq Q$ is the set of ***accept states***.

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of ***states***,
2. Σ is a finite set called the ***input alphabet***,
3. Γ is a finite set called the ***stack alphabet***,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to (Q \times \Gamma_\epsilon) \cup \{\emptyset\}$ is the ***transition function***,
5. $q_0 \in Q$ is the ***start state***, and
6. $F \subseteq Q$ is the set of ***accept states***.

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple

$(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

The transition function δ must satisfy the following condition:

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

The transition function δ must satisfy the following condition:

For every $q \in Q$, $a \in \Sigma$ and $x \in \Gamma$, exactly one of the values

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

The transition function δ must satisfy the following condition:

For every $q \in Q$, $a \in \Sigma$ and $x \in \Gamma$, exactly one of the values

- $\delta(q, a, x)$,

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

The transition function δ must satisfy the following condition:

For every $q \in Q$, $a \in \Sigma$ and $x \in \Gamma$, exactly one of the values

- $\delta(q, a, x)$,
- $\delta(q, a, \epsilon)$,

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

The transition function δ must satisfy the following condition:

For every $q \in Q$, $a \in \Sigma$ and $x \in \Gamma$, exactly one of the values

- $\delta(q, a, x)$,
- $\delta(q, a, \epsilon)$,
- $\delta(q, \epsilon, x)$,

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

The transition function δ must satisfy the following condition:

For every $q \in Q$, $a \in \Sigma$ and $x \in \Gamma$, exactly one of the values

- $\delta(q, a, x)$,
- $\delta(q, a, \epsilon)$,
- $\delta(q, \epsilon, x)$,
- $\delta(q, \epsilon, \epsilon)$

Deterministic Context-Free Languages

Definition (DPDA (确定型下推自动机))

A ***deterministic pushdown automaton*** (DPDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

The transition function δ must satisfy the following condition:

For every $q \in Q$, $a \in \Sigma$ and $x \in \Gamma$, exactly one of the values

- $\delta(q, a, x)$,
- $\delta(q, a, \epsilon)$,
- $\delta(q, \epsilon, x)$,
- $\delta(q, \epsilon, \epsilon)$

is not $\emptyset$.

Deterministic Context-Free Languages

Theorem

The class of DCFLs is closed under complementation.

Deterministic Context-Free Languages

Theorem

The class of DCFLs is closed under complementation.

This theorem implies that some CFLs are not DCFLs. Any CFL whose complement isn't a CFL isn't a DCFL.

Deterministic Context-Free Languages

Theorem

The class of DCFLs is closed under complementation.

This theorem implies that some CFLs are not DCFLs. Any CFL whose complement isn't a CFL isn't a DCFL.

Example

$A = \{a^i b^j c^k | i \neq j$ or $j \neq k$ where $i, j, k \geq 0\}$ is a CFL but not a DCFL.