

CS 373: Theory of Computation

Manoj Prabhakaran Mahesh Viswanathan

University of Illinois, Urbana-Champaign

Fall 2008

Part I

Computation of a PDA

Instantaneous Description

In order to describe a machine's execution, we need to capture a "snapshot" of the machine that completely determines future behavior

Instantaneous Description

In order to describe a machine's execution, we need to capture a "snapshot" of the machine that completely determines future behavior

- In the case of an NFA (or DFA), it is the state

Instantaneous Description

In order to describe a machine's execution, we need to capture a "snapshot" of the machine that completely determines future behavior

- In the case of an NFA (or DFA), it is the state
- In the case of a PDA, it is the state + stack contents

Instantaneous Description

In order to describe a machine's execution, we need to capture a "snapshot" of the machine that completely determines future behavior

- In the case of an NFA (or DFA), it is the state
- In the case of a PDA, it is the state + stack contents

Definition

An **instantaneous description** of a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is a pair $\langle q, \sigma \rangle$, where $q \in Q$ and $\sigma \in \Gamma^*$

Computation

Definition

For a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, string $w \in \Sigma^*$, and instantaneous descriptions $\langle q_1, \sigma_1 \rangle$ and $\langle q_2, \sigma_2 \rangle$, we say $\langle q_1, \sigma_1 \rangle \xrightarrow{w}_P \langle q_2, \sigma_2 \rangle$ iff there is a sequence of instantaneous descriptions $\langle r_0, s_0 \rangle, \langle r_1, s_1 \rangle, \dots, \langle r_k, s_k \rangle$ and a sequence x_1, x_2, \dots, x_k , where for each i , $x_i \in \Sigma \cup \{\epsilon\}$, such that

- $w = x_1 x_2 \cdots x_k$,

Computation

Definition

For a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, string $w \in \Sigma^*$, and instantaneous descriptions $\langle q_1, \sigma_1 \rangle$ and $\langle q_2, \sigma_2 \rangle$, we say $\langle q_1, \sigma_1 \rangle \xrightarrow{w}_P \langle q_2, \sigma_2 \rangle$ iff there is a sequence of instantaneous descriptions $\langle r_0, s_0 \rangle, \langle r_1, s_1 \rangle, \dots, \langle r_k, s_k \rangle$ and a sequence x_1, x_2, \dots, x_k , where for each i , $x_i \in \Sigma \cup \{\epsilon\}$, such that

- $w = x_1 x_2 \cdots x_k$,
- $r_0 = q_1$, and $s_0 = \sigma_1$,

Computation

Definition

For a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, string $w \in \Sigma^*$, and instantaneous descriptions $\langle q_1, \sigma_1 \rangle$ and $\langle q_2, \sigma_2 \rangle$, we say $\langle q_1, \sigma_1 \rangle \xrightarrow{w}_P \langle q_2, \sigma_2 \rangle$ iff there is a sequence of instantaneous descriptions $\langle r_0, s_0 \rangle, \langle r_1, s_1 \rangle, \dots, \langle r_k, s_k \rangle$ and a sequence x_1, x_2, \dots, x_k , where for each i , $x_i \in \Sigma \cup \{\epsilon\}$, such that

- $w = x_1 x_2 \cdots x_k$,
- $r_0 = q_1$, and $s_0 = \sigma_1$,
- $r_k = q_2$, and $s_k = \sigma_2$,

Computation

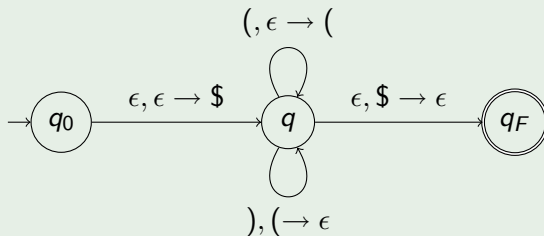
Definition

For a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, string $w \in \Sigma^*$, and instantaneous descriptions $\langle q_1, \sigma_1 \rangle$ and $\langle q_2, \sigma_2 \rangle$, we say $\langle q_1, \sigma_1 \rangle \xrightarrow{w}_P \langle q_2, \sigma_2 \rangle$ iff there is a sequence of instantaneous descriptions $\langle r_0, s_0 \rangle, \langle r_1, s_1 \rangle, \dots, \langle r_k, s_k \rangle$ and a sequence x_1, x_2, \dots, x_k , where for each i , $x_i \in \Sigma \cup \{\epsilon\}$, such that

- $w = x_1 x_2 \cdots x_k$,
- $r_0 = q_1$, and $s_0 = \sigma_1$,
- $r_k = q_2$, and $s_k = \sigma_2$,
- for every i , $(r_{i+1}, b) \in \delta(r_i, x_{i+1}, a)$ such that $s_i = as$ and $s_{i+1} = bs$, where $a, b \in \Gamma \cup \{\epsilon\}$ and $s \in \Gamma^*$

Example of Computation

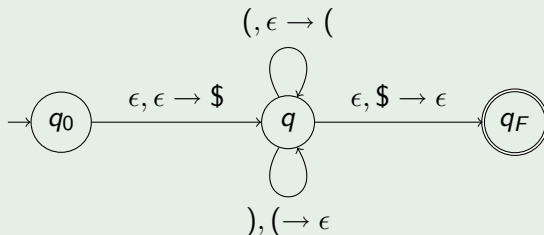
Example



$\langle q_0, \epsilon \rangle \xrightarrow{((}$ $\langle q, ((\$) \rangle$ because

Example of Computation

Example



$\langle q_0, \epsilon \rangle \xrightarrow{((() (} \langle q, ((\$) \rangle$ because

$\langle q_0, \epsilon \rangle \xrightarrow{x_1=\epsilon} \langle q, \$ \rangle \xrightarrow{x_2=(} \langle q, (\$) \rangle \xrightarrow{x_3=(} \langle q, ((\$) \rangle \xrightarrow{x_4=)} \langle q, (\$) \rangle \xrightarrow{x_5=(} \langle q, ((\$) \rangle$

Acceptance/Recognition

Definition

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ **accepts** a string $w \in \Sigma^*$ iff

Acceptance/Recognition

Definition

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ **accepts** a string $w \in \Sigma^*$ iff for some $q \in F$ and $\sigma \in \Gamma^*$, $\langle q_0, \epsilon \rangle \xrightarrow{w}_P \langle q, \sigma \rangle$

Acceptance/Recognition

Definition

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ **accepts** a string $w \in \Sigma^*$ iff for some $q \in F$ and $\sigma \in \Gamma^*$, $\langle q_0, \epsilon \rangle \xrightarrow{w}_P \langle q, \sigma \rangle$

Definition

The **language recognized/accepted** by a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is $L(P) = \{w \in \Sigma^* \mid P \text{ accepts } w\}$.

Acceptance/Recognition

Definition

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ **accepts** a string $w \in \Sigma^*$ iff for some $q \in F$ and $\sigma \in \Gamma^*$, $\langle q_0, \epsilon \rangle \xrightarrow{w}_P \langle q, \sigma \rangle$

Definition

The **language recognized/accepted** by a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is $L(P) = \{w \in \Sigma^* \mid P \text{ accepts } w\}$. A language L is said to be **accepted/recognized** by P if $L = L(P)$.

Part II

Equivalence of CFGs and PDAs

Expressive Power of CFGs and PDAs

We will show that CFGs and PDAs have equivalent expressive powers. More formally, . . .

Expressive Power of CFGs and PDAs

We will show that CFGs and PDAs have equivalent expressive powers. More formally, ...

Theorem

For every CFG G , there is a PDA P such that $L(G) = L(P)$. In addition, for every PDA P , there is a CFG G such that $L(P) = L(G)$.

Expressive Power of CFGs and PDAs

We will show that CFGs and PDAs have equivalent expressive powers. More formally, ...

Theorem

For every CFG G , there is a PDA P such that $L(G) = L(P)$. In addition, for every PDA P , there is a CFG G such that $L(P) = L(G)$. Thus, L is context-free iff there is a PDA P such that $L = L(P)$.

From CFG to PDA

Problem

Given grammar $G = (V, \Sigma, R, S)$ need to design PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that $L(P) = L(G)$.

From CFG to PDA

Problem

Given grammar $G = (V, \Sigma, R, S)$ need to design PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that $L(P) = L(G)$. In other words, given $w \in \Sigma^*$, the PDA P needs to figure out whether $S \xRightarrow{*}_G w$.

From CFG to PDA

Problem

Given grammar $G = (V, \Sigma, R, S)$ need to design PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that $L(P) = L(G)$. In other words, given $w \in \Sigma^*$, the PDA P needs to figure out whether $S \xRightarrow{*}_G w$.

Intuition

The PDA P will try to construct a derivation of w from S , by “applying” one rule at a time starting from S .

Simulating Derivations

Challenge I: Choosing rule to apply

How do you choose the rule of grammar to apply to get the next step of derivation?

Simulating Derivations

Challenge I: Choosing rule to apply

How do you choose the rule of grammar to apply to get the next step of derivation?

Use non-determinism to guess this choice!

Simulating Derivations

Challenge II: Storing Intermediate strings

In order to construct the derivation, we need to know what the current **intermediate string** is, so that we know what rules/steps can be applied next. How do we store this intermediate string?

Simulating Derivations

Challenge II: Storing Intermediate strings

In order to construct the derivation, we need to know what the current **intermediate string** is, so that we know what rules/steps can be applied next. How do we store this intermediate string?

- Store the intermediate string on the stack!

Simulating Derivations

Challenge II: Storing Intermediate strings

In order to construct the derivation, we need to know what the current **intermediate string** is, so that we know what rules/steps can be applied next. How do we store this intermediate string?

- Store the intermediate string on the stack!
 - Doesn't work!

Simulating Derivations

Challenge II: Storing Intermediate strings

In order to construct the derivation, we need to know what the current **intermediate string** is, so that we know what rules/steps can be applied next. How do we store this intermediate string?

- Store the intermediate string on the stack!
 - Doesn't work! The PDA can only read the top of the stack (which is say the leftmost symbol of the intermediate string), but it needs to know some variable of the string to know which rule to apply.

Simulating Derivations

Challenge II: Storing Intermediate strings

In order to construct the derivation, we need to know what the current **intermediate string** is, so that we know what rules/steps can be applied next. How do we store this intermediate string?

- Store the intermediate string on the stack!
 - Doesn't work! The PDA can only read the top of the stack (which is say the leftmost symbol of the intermediate string), but it needs to know some variable of the string to know which rule to apply.
- Store only part of the string on the stack

Simulating Derivations

Challenge II: Storing Intermediate strings

In order to construct the derivation, we need to know what the current **intermediate string** is, so that we know what rules/steps can be applied next. How do we store this intermediate string?

- Store the intermediate string on the stack!
 - Doesn't work! The PDA can only read the top of the stack (which is say the leftmost symbol of the intermediate string), but it needs to know some variable of the string to know which rule to apply.
- Store only part of the string on the stack; the part that immediately follows the first (leftmost) variable of the intermediate string

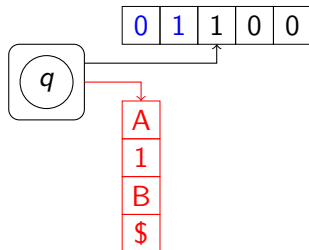
Simulating Derivations

Challenge II: Storing Intermediate strings

In order to construct the derivation, we need to know what the current **intermediate string** is, so that we know what rules/steps can be applied next. How do we store this intermediate string?

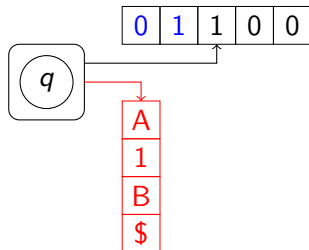
- Store the intermediate string on the stack!
 - Doesn't work! The PDA can only read the top of the stack (which is say the leftmost symbol of the intermediate string), but it needs to know some variable of the string to know which rule to apply.
- Store only part of the string on the stack; the part that immediately follows the first (leftmost) variable of the intermediate string
- Portion before the first variable in the intermediate string will be "matched" with the input

Example



The above PDA snapshot depicts intermediate string $01A1B$ as follows:

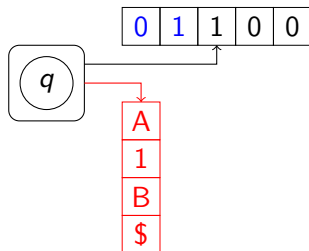
Example



The above PDA snapshot depicts intermediate string `01A1B` as follows:

- `01` has been read from the input

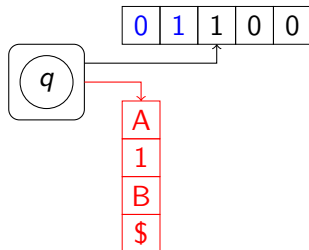
Example



The above PDA snapshot depicts intermediate string $01A1B$ as follows:

- 01 has been read from the input
- $A1B$ is on the stack;

Example



The above PDA snapshot depicts intermediate string $01A1B$ as follows:

- 01 has been read from the input
- $A1B$ is on the stack; “bottom of stack” is $\$$

PDA Algorithm

- 1 Push $\$$ and the start symbol onto the stack.

PDA Algorithm

- 1 Push $\$$ and the start symbol onto the stack.
- 2 Repeat the following steps

PDA Algorithm

- ① Push $\$$ and the start symbol onto the stack.
- ② Repeat the following steps
 - ① If top of the stack is a variable A , pick (nondeterministically) a rule for A , and push onto the stack the RHS of the rule; don't read any input symbol.

PDA Algorithm

- ① Push $\$$ and the start symbol onto the stack.
- ② Repeat the following steps
 - ① If top of the stack is a variable A , pick (nondeterministically) a rule for A , and push onto the stack the RHS of the rule; don't read any input symbol.
 - ② If top of the stack is a terminal a , then read the next input symbol. If the input symbol is not a , then this branch dies. Otherwise, continue.

PDA Algorithm

- ① Push $\$$ and the start symbol onto the stack.
- ② Repeat the following steps
 - ① If top of the stack is a variable A , pick (nondeterministically) a rule for A , and push onto the stack the RHS of the rule; don't read any input symbol.
 - ② If top of the stack is a terminal a , then read the next input symbol. If the input symbol is not a , then this branch dies. Otherwise, continue.
 - ③ If top of the stack is $\$$ then pop the symbol and go to accepting state. There are no transition from accept state, and so if input is accepted only if there are no more input symbols.

Pushing Multiple Symbols

PDA Transitions

Formally, $\delta(q, x, a)$ contains pairs (q', b) which means that the PDA reads at most one symbol from input (if $x \neq \epsilon$), pops the top of the stack (if $a \neq \epsilon$) and pushes at most one symbol onto the stack (if $b \neq \epsilon$).

Pushing Multiple Symbols

PDA Transitions

Formally, $\delta(q, x, a)$ contains pairs (q', b) which means that the PDA reads at most one symbol from input (if $x \neq \epsilon$), pops the top of the stack (if $a \neq \epsilon$) and pushes at most one symbol onto the stack (if $b \neq \epsilon$).

To simplify the construction, we will assume that we can push as many symbols as we want in one step. Thus, $\delta(q, x, a)$ contains pairs (q', σ) , where $\sigma = b_1 b_2 \cdots b_k \in \Gamma^*$

Pushing Multiple Symbols

PDA Transitions

Formally, $\delta(q, x, a)$ contains pairs (q', b) which means that the PDA reads at most one symbol from input (if $x \neq \epsilon$), pops the top of the stack (if $a \neq \epsilon$) and pushes at most one symbol onto the stack (if $b \neq \epsilon$).

To simplify the construction, we will assume that we can push as many symbols as we want in one step. Thus, $\delta(q, x, a)$ contains pairs (q', σ) , where $\sigma = b_1 b_2 \cdots b_k \in \Gamma^*$

- Can easily be carried out by additional states that push the symbols of σ one by one.

Pushing Multiple Symbols

PDA Transitions

Formally, $\delta(q, x, a)$ contains pairs (q', b) which means that the PDA reads at most one symbol from input (if $x \neq \epsilon$), pops the top of the stack (if $a \neq \epsilon$) and pushes at most one symbol onto the stack (if $b \neq \epsilon$).

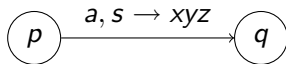
To simplify the construction, we will assume that we can push as many symbols as we want in one step. Thus, $\delta(q, x, a)$ contains pairs (q', σ) , where $\sigma = b_1 b_2 \cdots b_k \in \Gamma^*$

- Can easily be carried out by additional states that push the symbols of σ one by one. Formally, we have new states q_1, \dots, q_{k-1} and transitions

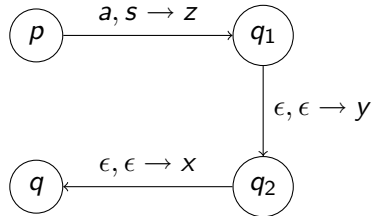
$$\begin{aligned} (q_1, b_k) &\in \delta(q, x, a), \quad \delta(q_1, \epsilon, \epsilon) = \{(q_2, b_{k-1})\}, \\ \delta(q_2, \epsilon, \epsilon) &= \{(q_3, b_{k-2})\}, \quad \dots \quad \delta(q_{k-1}, \epsilon, \epsilon) = \{(q', b_1)\} \end{aligned}$$

Pushing Multiple Symbols

An Example



Transition pushing multiple symbols



Multiple pushes implemented by single pushes

Formal Definition of PDA

Let $G = (V, \Sigma, R, S)$ be a CFG. Define $P_G = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that

Formal Definition of PDA

Let $G = (V, \Sigma, R, S)$ be a CFG. Define $P_G = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that

- $Q = \{q_s, q_\ell, q_a\}$

Formal Definition of PDA

Let $G = (V, \Sigma, R, S)$ be a CFG. Define $P_G = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that

- $Q = \{q_s, q_\ell, q_a\}$
- $\Gamma = V \cup \Sigma$

Formal Definition of PDA

Let $G = (V, \Sigma, R, S)$ be a CFG. Define $P_G = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that

- $Q = \{q_s, q_\ell, q_a\}$
- $\Gamma = V \cup \Sigma$
- $q_0 = q_s$

Formal Definition of PDA

Let $G = (V, \Sigma, R, S)$ be a CFG. Define $P_G = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that

- $Q = \{q_s, q_\ell, q_a\}$
- $\Gamma = V \cup \Sigma$
- $q_0 = q_s$
- $F = \{q_a\}$

Formal Definition of PDA

Let $G = (V, \Sigma, R, S)$ be a CFG. Define $P_G = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that

- $Q = \{q_s, q_\ell, q_a\}$
- $\Gamma = V \cup \Sigma$
- $q_0 = q_s$
- $F = \{q_a\}$
- Assuming $a \in \Sigma$ and $A \in V$, δ is given by

$$\delta(q_s, \epsilon, \epsilon) = \{(q_\ell, S\$)\}$$

Formal Definition of PDA

Let $G = (V, \Sigma, R, S)$ be a CFG. Define $P_G = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that

- $Q = \{q_s, q_\ell, q_a\}$
- $\Gamma = V \cup \Sigma$
- $q_0 = q_s$
- $F = \{q_a\}$
- Assuming $a \in \Sigma$ and $A \in V$, δ is given by

$$\delta(q_s, \epsilon, \epsilon) = \{(q_\ell, S\$)\}$$

$$\delta(q_\ell, a, a) = \{(q_\ell, \epsilon)\}$$

Formal Definition of PDA

Let $G = (V, \Sigma, R, S)$ be a CFG. Define $P_G = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that

- $Q = \{q_s, q_\ell, q_a\}$
- $\Gamma = V \cup \Sigma$
- $q_0 = q_s$
- $F = \{q_a\}$
- Assuming $a \in \Sigma$ and $A \in V$, δ is given by

$$\delta(q_s, \epsilon, \epsilon) = \{(q_\ell, S\$)\}$$

$$\delta(q_\ell, a, a) = \{(q_\ell, \epsilon)\}$$

$$\delta(q_\ell, \epsilon, A) = \{(q_\ell, w) \mid A \rightarrow w \in R\}$$

Formal Definition of PDA

Let $G = (V, \Sigma, R, S)$ be a CFG. Define $P_G = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that

- $Q = \{q_s, q_\ell, q_a\}$
- $\Gamma = V \cup \Sigma$
- $q_0 = q_s$
- $F = \{q_a\}$
- Assuming $a \in \Sigma$ and $A \in V$, δ is given by

$$\begin{aligned}\delta(q_s, \epsilon, \epsilon) &= \{(q_\ell, S\$)\} & \delta(q_\ell, a, a) &= \{(q_\ell, \epsilon)\} \\ \delta(q_\ell, \epsilon, A) &= \{(q_\ell, w) \mid A \rightarrow w \in R\} & \delta(q_\ell, \epsilon, \$) &= \{(q_a, \epsilon)\}\end{aligned}$$

Formal Definition of PDA

Let $G = (V, \Sigma, R, S)$ be a CFG. Define $P_G = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that

- $Q = \{q_s, q_\ell, q_a\}$
- $\Gamma = V \cup \Sigma$
- $q_0 = q_s$
- $F = \{q_a\}$
- Assuming $a \in \Sigma$ and $A \in V$, δ is given by

$$\begin{aligned} \delta(q_s, \epsilon, \epsilon) &= \{(q_\ell, S\$)\} & \delta(q_\ell, a, a) &= \{(q_\ell, \epsilon)\} \\ \delta(q_\ell, \epsilon, A) &= \{(q_\ell, w) \mid A \rightarrow w \in R\} & \delta(q_\ell, \epsilon, \$) &= \{(q_a, \epsilon)\} \end{aligned}$$

In all other cases, $\delta(\cdot) = \emptyset$

Formal Definition of PDA

Let $G = (V, \Sigma, R, S)$ be a CFG. Define $P_G = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that

- $Q = \{q_s, q_\ell, q_a\}$
- $\Gamma = V \cup \Sigma$
- $q_0 = q_s$
- $F = \{q_a\}$
- Assuming $a \in \Sigma$ and $A \in V$, δ is given by

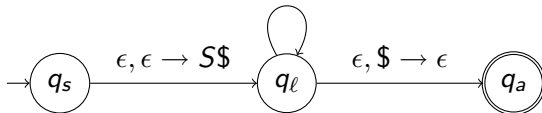
$$\begin{aligned} \delta(q_s, \epsilon, \epsilon) &= \{(q_\ell, S\$)\} & \delta(q_\ell, a, a) &= \{(q_\ell, \epsilon)\} \\ \delta(q_\ell, \epsilon, A) &= \{(q_\ell, w) \mid A \rightarrow w \in R\} & \delta(q_\ell, \epsilon, \$) &= \{(q_a, \epsilon)\} \end{aligned}$$

In all other cases, $\delta(\cdot) = \emptyset$

To remove steps that push more than one symbol, we will add more states.

State Diagram of PDA

$\epsilon, A \rightarrow w$ for $A \rightarrow w \in R$
 $a, a \rightarrow \epsilon$ for $a \in \Sigma$



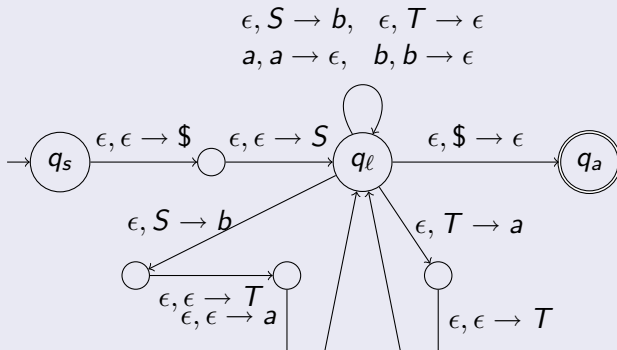
State Diagram of P_G

CFG to PDA

An example

Consider the CFG

$G = (\{S, T\}, \{a, b\}, \{S \rightarrow aTb \mid b, T \rightarrow Ta \mid \epsilon\}, S)$. The PDA P_G is as follows.



Leftmost Derivations

Derivations

At each step, replace *some* variable in the intermediate string by the right-hand side of a rule in the grammar.

Leftmost Derivations

Derivations

At each step, replace *some* variable in the intermediate string by the right-hand side of a rule in the grammar.

Leftmost Derivations

At each step, replace the *leftmost* variable in the intermediate string by the right-hand side of a rule in the grammar.

Leftmost Derivations

Derivations

At each step, replace *some* variable in the intermediate string by the right-hand side of a rule in the grammar.

Leftmost Derivations

At each step, replace the *leftmost* variable in the intermediate string by the right-hand side of a rule in the grammar.

We will say $\alpha \Rightarrow_{lm} \beta$ if β is obtained from α by a leftmost derivation step.

Head/Tail of Intermediate Strings

Definition

Let $\beta = wA\alpha \in (V \cup \Sigma)^*$ be such that $w \in \Sigma^*$, $A \in V$ and $\alpha \in (V \cup \Sigma)^*$.

Head/Tail of Intermediate Strings

Definition

Let $\beta = wA\alpha \in (V \cup \Sigma)^*$ be such that $w \in \Sigma^*$, $A \in V$ and $\alpha \in (V \cup \Sigma)^*$. The **head** of β is w , and the **tail** is $A\alpha$.

Head/Tail of Intermediate Strings

Definition

Let $\beta = wA\alpha \in (V \cup \Sigma)^*$ be such that $w \in \Sigma^*$, $A \in V$ and $\alpha \in (V \cup \Sigma)^*$. The **head** of β is w , and the **tail** is $A\alpha$.

Example

The head of string $01A1B$ is 01 and the tail is $A1B$.

Correctness of Construction

Proposition

Let G be a CFG and let P_G be the PDA constructed. Then $L(P_G) = L(G)$

Proof.

The proof of correctness is based on the following two observations.

- Let $S \xRightarrow{*}_{\text{lm}} x\alpha$ be a leftmost derivation, where x is the head and α the tail of the string $x\alpha$. Then, $\langle q_s, \epsilon \rangle \xrightarrow{x}_{P_G} \langle q_l, \alpha \$ \rangle$.

Correctness of Construction

Proposition

Let G be a CFG and let P_G be the PDA constructed. Then $L(P_G) = L(G)$

Proof.

The proof of correctness is based on the following two observations.

- Let $S \xRightarrow{*}_{\text{lm}} x\alpha$ be a leftmost derivation, where x is the head and α the tail of the string $x\alpha$. Then, $\langle q_s, \epsilon \rangle \xrightarrow{x}_{P_G} \langle q_l, \alpha\$ \rangle$.
Proved by induction on the number of leftmost derivation steps.

Correctness of Construction

Proposition

Let G be a CFG and let P_G be the PDA constructed. Then $L(P_G) = L(G)$

Proof.

The proof of correctness is based on the following two observations.

- Let $S \xRightarrow{*}_{\text{lm}} x\alpha$ be a leftmost derivation, where x is the head and α the tail of the string $x\alpha$. Then, $\langle q_s, \epsilon \rangle \xrightarrow{x}_{P_G} \langle q_l, \alpha\$ \rangle$.
Proved by induction on the number of leftmost derivation steps.
- For every $A \in V$, if $\langle q_l, A \rangle \xrightarrow{x}_{P_G} \langle q_l, \epsilon \rangle$ then $A \xRightarrow{*} x$.

Correctness of Construction

Proposition

Let G be a CFG and let P_G be the PDA constructed. Then $L(P_G) = L(G)$

Proof.

The proof of correctness is based on the following two observations.

- Let $S \xRightarrow{*}_{\text{lm}} x\alpha$ be a leftmost derivation, where x is the head and α the tail of the string $x\alpha$. Then, $\langle q_s, \epsilon \rangle \xrightarrow{x}_{P_G} \langle q_l, \alpha\$ \rangle$. Proved by induction on the number of leftmost derivation steps.
- For every $A \in V$, if $\langle q_l, A \rangle \xrightarrow{x}_{P_G} \langle q_l, \epsilon \rangle$ then $A \xRightarrow{*} x$. Proved by induction on the number of steps taken by P_G .

Correctness of Construction

Proposition

Let G be a CFG and let P_G be the PDA constructed. Then $L(P_G) = L(G)$

Proof.

The proof of correctness is based on the following two observations.

- Let $S \xRightarrow{*}_{\text{lm}} x\alpha$ be a leftmost derivation, where x is the head and α the tail of the string $x\alpha$. Then, $\langle q_s, \epsilon \rangle \xrightarrow{x}_{P_G} \langle q_l, \alpha\$ \rangle$. Proved by induction on the number of leftmost derivation steps.
- For every $A \in V$, if $\langle q_l, A \rangle \xrightarrow{x}_{P_G} \langle q_l, \epsilon \rangle$ then $A \xRightarrow{*} x$. Proved by induction on the number of steps taken by P_G .

Proof is skipped; see notes.



From PDA to CFG

Proposition

For any PDA P , there is a CFG G such that $L(P) = L(G)$.

From PDA to CFG

Proposition

For any PDA P , there is a CFG G such that $L(P) = L(G)$.

Proof Outline

From PDA to CFG

Proposition

For any PDA P , there is a CFG G such that $L(P) = L(G)$.

Proof Outline

- 1 For every PDA P there is a **normalized** PDA P_N such that $L(P) = L(P_N)$.

From PDA to CFG

Proposition

For any PDA P , there is a CFG G such that $L(P) = L(G)$.

Proof Outline

- 1 For every PDA P there is a **normalized** PDA P_N such that $L(P) = L(P_N)$.
- 2 For every normalized PDA P_N there is a CFG G such that $L(P_N) = L(G)$.

Normalized PDAs

Definition

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is **normalized** iff it satisfies the following conditions.

Normalized PDAs

Definition

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is **normalized** iff it satisfies the following conditions.

- Has exactly one accept state, i.e., $F = \{q_a\}$ for some $q_a \in Q$

Normalized PDAs

Definition

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is **normalized** iff it satisfies the following conditions.

- Has exactly one accept state, i.e., $F = \{q_a\}$ for some $q_a \in Q$
- Empties its stack before accepting, i.e., if $\langle q_0, \epsilon \rangle \xrightarrow{w}_P \langle q_a, \sigma \rangle$ then $\sigma = \epsilon$.

Normalized PDAs

Definition

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is **normalized** iff it satisfies the following conditions.

- Has exactly one accept state, i.e., $F = \{q_a\}$ for some $q_a \in Q$
- Empties its stack before accepting, i.e., if $\langle q_0, \epsilon \rangle \xrightarrow{w}_P \langle q_a, \sigma \rangle$ then $\sigma = \epsilon$.
- Each transition either pushes one symbol, or it pops one symbol. There are no transitions that both *push and pop*, nor transitions that leave the stack unaffected.

Normalizing a PDA

Proposition

For every PDA P , there is a normalized PDA P_N such that $L(P) = L(P_N)$

Normalizing a PDA

Proposition

For every PDA P , there is a normalized PDA P_N such that $L(P) = L(P_N)$

Proof Sketch

We will transform P in a series of steps, each time ensuring that the language does not change.

Normalizing a PDA

Proposition

For every PDA P , there is a normalized PDA P_N such that $L(P) = L(P_N)$

Proof Sketch

We will transform P in a series of steps, each time ensuring that the language does not change.

- We will ensure that there is only one accept state

Normalizing a PDA

Proposition

For every PDA P , there is a normalized PDA P_N such that $L(P) = L(P_N)$

Proof Sketch

We will transform P in a series of steps, each time ensuring that the language does not change.

- We will ensure that there is only one accept state
- Next, we will ensure that all symbols are popped before accept state is reached.

Normalizing a PDA

Proposition

For every PDA P , there is a normalized PDA P_N such that $L(P) = L(P_N)$

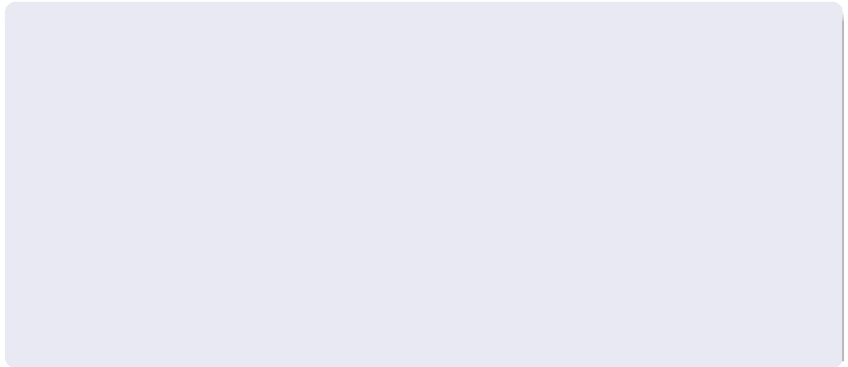
Proof Sketch

We will transform P in a series of steps, each time ensuring that the language does not change.

- We will ensure that there is only one accept state
- Next, we will ensure that all symbols are popped before accept state is reached.
- Finally, we will transform transitions to be either push or pop (not both or neither).

Normalizing a PDA

One accept state



Normalizing a PDA

One accept state

To ensure one accept state, add ϵ -transitions (which do not change the stack) from old accept states to **a new accept state**.

Normalizing a PDA

One accept state

To ensure one accept state, add ϵ -transitions (which do not change the stack) from old accept states to **a new accept state**.

Formally, given $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, let $P' = (Q', \Sigma, \Gamma, \delta', q_0, F')$ where

- $Q' = Q \cup \{q_a\}$, where $q_a \notin Q$
- $F' = \{q_a\}$
- $\delta'(q, x, a) = \delta(q, x, a)$ if $q \in Q \setminus F$ or $x \neq \epsilon$ or $a \neq \epsilon$, and $\delta'(q, \epsilon, \epsilon) = \delta(q, \epsilon, \epsilon) \cup \{(q_a, \epsilon)\}$ for $q \in F$, and $\delta'(q_a, x, a) = \emptyset$.

Normalizing a PDA

Emptying stack before acceptance

Normalizing a PDA

Emptying stack before acceptance

First push a new symbol $\$$ before starting computation, and from sole accept state, pop all symbols before popping $\$$ and moving to a new accept state.

Normalizing a PDA

Emptying stack before acceptance

First push a new symbol $\$$ before starting computation, and from sole accept state, pop all symbols before popping $\$$ and moving to a new accept state.

i.e., given $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$, let $P' = (Q', \Sigma, \Gamma', \delta', q'_0, F')$:

- $\Gamma' = \Gamma \cup \{\$\}$, where $\$ \notin \Gamma$
- $Q' = Q \cup \{q_i, q_p, q_a\}$ where $q_i, q_p, q_a \notin Q$
- $q'_0 = q_i$
- $F' = \{q_a\}$
- $\delta'(q, x, a) = \delta(q, x, a)$ for $q \in Q \setminus \{q_a\}$ or $x \neq \epsilon$ or $a \neq \epsilon$. For q_a , we have $\delta'(q_a, \epsilon, \epsilon) = \delta(q_a, \epsilon, \epsilon) \cup \{(q_p, \epsilon)\}$. In addition, we have $\delta'(q_i, \epsilon, \epsilon) = \{(q_0, \$)\}$, and $\delta'(q_p, \epsilon, a) = \{(q_p, \epsilon)\}$ for $a \in \Gamma$, and $\delta'(q_p, \epsilon, \$) = \{(q_a, \epsilon)\}$. In all other cases δ' is \emptyset .

Normalizing a PDA

Only pushes or pops

There are two kinds of transitions that need fixing

Normalizing a PDA

Only pushes or pops

There are two kinds of transitions that need fixing

- Transition of the form $q \xrightarrow{x, a \rightarrow b} q'$, where $a, b \in \Gamma$, i.e., those that push and pop in one step

Normalizing a PDA

Only pushes or pops

There are two kinds of transitions that need fixing

- Transition of the form $q \xrightarrow{x, a \rightarrow b} q'$, where $a, b \in \Gamma$, i.e., those that push and pop in one step
 - Replace this by two steps, where you first pop a and then push b : $q \xrightarrow{x, a \rightarrow \epsilon} q'' \xrightarrow{\epsilon, \epsilon \rightarrow b} q'$. (q'' is a new state involved in only these transitions.)

Normalizing a PDA

Only pushes or pops

There are two kinds of transitions that need fixing

- Transition of the form $q \xrightarrow{x, a \rightarrow b} q'$, where $a, b \in \Gamma$, i.e., those that push and pop in one step
 - Replace this by two steps, where you first pop a and then push b : $q \xrightarrow{x, a \rightarrow \epsilon} q'' \xrightarrow{\epsilon, \epsilon \rightarrow b} q'$. (q'' is a new state involved in only these transitions.)
- Transition of the form $q \xrightarrow{x, \epsilon \rightarrow \epsilon} q'$, i.e., those that neither push nor pop

Normalizing a PDA

Only pushes or pops

There are two kinds of transitions that need fixing

- Transition of the form $q \xrightarrow{x,a \rightarrow b} q'$, where $a, b \in \Gamma$, i.e., those that push and pop in one step
 - Replace this by two steps, where you first pop a and then push b : $q \xrightarrow{x,a \rightarrow \epsilon} q'' \xrightarrow{\epsilon, \epsilon \rightarrow b} q'$. (q'' is a new state involved in only these transitions.)
- Transition of the form $q \xrightarrow{x, \epsilon \rightarrow \epsilon} q'$, i.e., those that neither push nor pop
 - Replace this by two steps, where first a dummy symbol is pushed, and then in the second step the dummy symbol is popped: $q \xrightarrow{x, \epsilon \rightarrow \partial} q'' \xrightarrow{\epsilon, \partial \rightarrow \epsilon} q'$. (q'' is a new state involved in only these transitions.)

Normalizing a PDA

Only pushes or pops

There are two kinds of transitions that need fixing

- Transition of the form $q \xrightarrow{x,a \rightarrow b} q'$, where $a, b \in \Gamma$, i.e., those that push and pop in one step
 - Replace this by two steps, where you first pop a and then push b : $q \xrightarrow{x,a \rightarrow \epsilon} q'' \xrightarrow{\epsilon, \epsilon \rightarrow b} q'$. (q'' is a new state involved in only these transitions.)
- Transition of the form $q \xrightarrow{x, \epsilon \rightarrow \epsilon} q'$, i.e., those that neither push nor pop
 - Replace this by two steps, where first a dummy symbol is pushed, and then in the second step the dummy symbol is popped: $q \xrightarrow{x, \epsilon \rightarrow \partial} q'' \xrightarrow{\epsilon, \partial \rightarrow \epsilon} q'$. (q'' is a new state involved in only these transitions.)

Normalizing a PDA

Only pushes or pops

There are two kinds of transitions that need fixing

- Transition of the form $q \xrightarrow{x,a \rightarrow b} q'$, where $a, b \in \Gamma$, i.e., those that push and pop in one step
 - Replace this by two steps, where you first pop a and then push b : $q \xrightarrow{x,a \rightarrow \epsilon} q'' \xrightarrow{\epsilon, \epsilon \rightarrow b} q'$. (q'' is a new state involved in only these transitions.)
- Transition of the form $q \xrightarrow{x, \epsilon \rightarrow \epsilon} q'$, i.e., those that neither push nor pop
 - Replace this by two steps, where first a dummy symbol is pushed, and then in the second step the dummy symbol is popped: $q \xrightarrow{x, \epsilon \rightarrow \partial} q'' \xrightarrow{\epsilon, \partial \rightarrow \epsilon} q'$. (q'' is a new state involved in only these transitions.)

(Formal definition skipped.)

CFGs for Normalized PDAs

Intuitions

- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$ be a normalized PDA. $w \in L(P)$ iff $\langle q_0, \epsilon \rangle \xrightarrow{w}_P \langle q_a, \epsilon \rangle$.

CFGs for Normalized PDAs

Intuitions

- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$ be a normalized PDA. $w \in L(P)$ iff $\langle q_0, \epsilon \rangle \xrightarrow{w}_P \langle q_a, \epsilon \rangle$.
- If, for every $p, q \in Q$, we can describe $L_{p,q} = \{w \mid \langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle\}$ using a CFG, then we are done because $L(P)$ is nothing but L_{q_0, q_a} .

CFGs for Normalized PDAs

Intuitions

- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$ be a normalized PDA. $w \in L(P)$ iff $\langle q_0, \epsilon \rangle \xrightarrow{w}_P \langle q_a, \epsilon \rangle$.
- If, for every $p, q \in Q$, we can describe $L_{p,q} = \{w \mid \langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle\}$ using a CFG, then we are done because $L(P)$ is nothing but L_{q_0, q_a} .
- So CFG will have variables $A_{p,q}$ such that $A_{p,q} \xRightarrow{*} w$ iff $w \in L_{p,q}$.

CFGs for Normalized PDAs

Intuitions

- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$ be a normalized PDA. $w \in L(P)$ iff $\langle q_0, \epsilon \rangle \xrightarrow{w}_P \langle q_a, \epsilon \rangle$.
- If, for every $p, q \in Q$, we can describe $L_{p,q} = \{w \mid \langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle\}$ using a CFG, then we are done because $L(P)$ is nothing but L_{q_0, q_a} .
- So CFG will have variables $A_{p,q}$ such that $A_{p,q} \xRightarrow{*} w$ iff $w \in L_{p,q}$.
- What are the rules for $A_{p,q}$?

Rules for the grammar

Consider $w \in L_{p,q}$, and a computation corresponding to $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$.

Rules for the grammar

Consider $w \in L_{p,q}$, and a computation corresponding to $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$. Since the computation starts with empty stack, the first step must be a push, and last step must be a pop, since we end with empty stack.

Rules for the grammar

Consider $w \in L_{p,q}$, and a computation corresponding to $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$. Since the computation starts with empty stack, the first step must be a push, and last step must be a pop, since we end with empty stack.

- **Case I:** The first symbol pushed is popped only at the end.

Rules for the grammar

Consider $w \in L_{p,q}$, and a computation corresponding to $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$. Since the computation starts with empty stack, the first step must be a push, and last step must be a pop, since we end with empty stack.

- **Case I:** The first symbol pushed is popped only at the end. So we have $\langle p, \epsilon \rangle \xrightarrow{a} \langle r, A \rangle \xrightarrow{u} \langle s, A \rangle \xrightarrow{b} \langle q, \epsilon \rangle$, with $w = ab$.

Rules for the grammar

Consider $w \in L_{p,q}$, and a computation corresponding to $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$. Since the computation starts with empty stack, the first step must be a push, and last step must be a pop, since we end with empty stack.

- **Case I:** The first symbol pushed is popped only at the end. So we have $\langle p, \epsilon \rangle \xrightarrow{a} \langle r, A \rangle \xrightarrow{u} \langle s, A \rangle \xrightarrow{b} \langle q, \epsilon \rangle$, with $w = aub$. And $u \in L_{r,s}$.

Rules for the grammar

Consider $w \in L_{p,q}$, and a computation corresponding to $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$. Since the computation starts with empty stack, the first step must be a push, and last step must be a pop, since we end with empty stack.

- **Case I:** The first symbol pushed is popped only at the end. So we have $\langle p, \epsilon \rangle \xrightarrow{a} \langle r, A \rangle \xrightarrow{u} \langle s, A \rangle \xrightarrow{b} \langle q, \epsilon \rangle$, with $w = aub$. And $u \in L_{r,s}$. Can be captured by rule $A_{p,q} \rightarrow aA_{r,s}b$.

Rules for the grammar

Consider $w \in L_{p,q}$, and a computation corresponding to $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$. Since the computation starts with empty stack, the first step must be a push, and last step must be a pop, since we end with empty stack.

- **Case I:** The first symbol pushed is popped only at the end. So we have $\langle p, \epsilon \rangle \xrightarrow{a} \langle r, A \rangle \xrightarrow{u} \langle s, A \rangle \xrightarrow{b} \langle q, \epsilon \rangle$, with $w = aub$. And $u \in L_{r,s}$. Can be captured by rule $A_{p,q} \rightarrow aA_{r,s}b$.
- **Case II:** First symbol pushed is popped in the middle of computation (and then stack is empty).

Rules for the grammar

Consider $w \in L_{p,q}$, and a computation corresponding to $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$. Since the computation starts with empty stack, the first step must be a push, and last step must be a pop, since we end with empty stack.

- **Case I:** The first symbol pushed is popped only at the end. So we have $\langle p, \epsilon \rangle \xrightarrow{a} \langle r, A \rangle \xrightarrow{u} \langle s, A \rangle \xrightarrow{b} \langle q, \epsilon \rangle$, with $w = aub$. And $u \in L_{r,s}$. Can be captured by rule $A_{p,q} \rightarrow aA_{r,s}b$.
- **Case II:** First symbol pushed is popped in the middle of computation (and then stack is empty). So we have $\langle p, \epsilon \rangle \xrightarrow{u_1} \langle r, \epsilon \rangle \xrightarrow{u_2} \langle q, \epsilon \rangle$.

Rules for the grammar

Consider $w \in L_{p,q}$, and a computation corresponding to $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$. Since the computation starts with empty stack, the first step must be a push, and last step must be a pop, since we end with empty stack.

- **Case I:** The first symbol pushed is popped only at the end. So we have $\langle p, \epsilon \rangle \xrightarrow{a} \langle r, A \rangle \xrightarrow{u} \langle s, A \rangle \xrightarrow{b} \langle q, \epsilon \rangle$, with $w = aub$. And $u \in L_{r,s}$. Can be captured by rule $A_{p,q} \rightarrow aA_{r,s}b$.
- **Case II:** First symbol pushed is popped in the middle of computation (and then stack is empty). So we have $\langle p, \epsilon \rangle \xrightarrow{u_1} \langle r, \epsilon \rangle \xrightarrow{u_2} \langle q, \epsilon \rangle$. Can be captured by rule $A_{p,q} \rightarrow A_{p,r}A_{r,q}$

Formal Construction

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$ be a normalized PDA. Define $G_P = (V, \Sigma, R, S)$ where

Formal Construction

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$ be a normalized PDA. Define $G_P = (V, \Sigma, R, S)$ where

- $V = \{A_{p,q} \mid p, q \in Q\},$

Formal Construction

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$ be a normalized PDA. Define $G_P = (V, \Sigma, R, S)$ where

- $V = \{A_{p,q} \mid p, q \in Q\},$
- $S = A_{q_0, q_a}$

Formal Construction

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$ be a normalized PDA. Define $G_P = (V, \Sigma, R, S)$ where

- $V = \{A_{p,q} \mid p, q \in Q\}$,
- $S = A_{q_0, q_a}$
- And the rules in R are

Formal Construction

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$ be a normalized PDA. Define $G_P = (V, \Sigma, R, S)$ where

- $V = \{A_{p,q} \mid p, q \in Q\}$,
- $S = A_{q_0, q_a}$
- And the rules in R are
 - For every $p \in Q$, $A_{p,p} \rightarrow \epsilon$

Formal Construction

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$ be a normalized PDA. Define $G_P = (V, \Sigma, R, S)$ where

- $V = \{A_{p,q} \mid p, q \in Q\}$,
- $S = A_{q_0, q_a}$
- And the rules in R are
 - For every $p \in Q$, $A_{p,p} \rightarrow \epsilon$
 - For every $p, q, r \in Q$, $A_{p,q} \rightarrow A_{p,r}A_{r,q}$

Formal Construction

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$ be a normalized PDA. Define $G_P = (V, \Sigma, R, S)$ where

- $V = \{A_{p,q} \mid p, q \in Q\}$,
- $S = A_{q_0, q_a}$
- And the rules in R are
 - For every $p \in Q$, $A_{p,p} \rightarrow \epsilon$
 - For every $p, q, r \in Q$, $A_{p,q} \rightarrow A_{p,r}A_{r,q}$
 - For every $p, q, r, s \in Q$, $\gamma \in \Gamma$, $a, b \in \Sigma \cup \{\epsilon\}$, if $(r, \gamma) \in \delta(p, a, \epsilon)$ and $(q, \epsilon) \in \delta(r, b, \gamma)$ then $A_{p,q} \rightarrow aA_{r,s}b$

Correctness of Construction

Proposition

Let P be a normalized PDA and let G_P be the corresponding CFG. Then $A_{p,q} \xRightarrow{*} w$ iff $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$.

Correctness of Construction

Proposition

Let P be a normalized PDA and let G_P be the corresponding CFG. Then $A_{p,q} \stackrel{*}{\Rightarrow} w$ iff $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$.

Proof.

The two directions are proved as follows

Correctness of Construction

Proposition

Let P be a normalized PDA and let G_P be the corresponding CFG. Then $A_{p,q} \xRightarrow{*} w$ iff $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$.

Proof.

The two directions are proved as follows

⇒ By induction on the number of steps in the derivation.

Correctness of Construction

Proposition

Let P be a normalized PDA and let G_P be the corresponding CFG. Then $A_{p,q} \xRightarrow{*} w$ iff $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$.

Proof.

The two directions are proved as follows

- \Rightarrow By induction on the number of steps in the derivation.
- \Leftarrow By induction on the number of steps in the computation.

Correctness of Construction

Proposition

Let P be a normalized PDA and let G_P be the corresponding CFG. Then $A_{p,q} \xRightarrow{*} w$ iff $\langle p, \epsilon \rangle \xrightarrow{w}_P \langle q, \epsilon \rangle$.

Proof.

The two directions are proved as follows

\Rightarrow By induction on the number of steps in the derivation.

\Leftarrow By induction on the number of steps in the computation.

Proof details in textbook. □

Tying all the Ends

Proposition

Let P be a PDA then $L(P)$ is context-free.

Proof.

Tying all the Ends

Proposition

Let P be a PDA then $L(P)$ is context-free.

Proof.

- A normalized PDA P_N can be constructed such that $L(P) = L(P_N)$

Tying all the Ends

Proposition

Let P be a PDA then $L(P)$ is context-free.

Proof.

- A normalized PDA P_N can be constructed such that $L(P) = L(P_N)$
- A grammar G_P can be constructed such that $L(G_P) = L(P_N) = L(P)$. This is because

Tying all the Ends

Proposition

Let P be a PDA then $L(P)$ is context-free.

Proof.

- A normalized PDA P_N can be constructed such that $L(P) = L(P_N)$
- A grammar G_P can be constructed such that $L(G_P) = L(P_N) = L(P)$. This is because
 - $S = A_{q_0, q_a} \xRightarrow{*} w$ iff

Tying all the Ends

Proposition

Let P be a PDA then $L(P)$ is context-free.

Proof.

- A normalized PDA P_N can be constructed such that $L(P) = L(P_N)$
- A grammar G_P can be constructed such that $L(G_P) = L(P_N) = L(P)$. This is because
 - $S = A_{q_0, q_a} \xRightarrow{*} w$ iff $\langle q_0, \epsilon \rangle \xrightarrow{w}_{P_N} \langle q_a, \epsilon \rangle$ (by previous proposition) iff

Tying all the Ends

Proposition

Let P be a PDA then $L(P)$ is context-free.

Proof.

- A normalized PDA P_N can be constructed such that $L(P) = L(P_N)$
- A grammar G_P can be constructed such that $L(G_P) = L(P_N) = L(P)$. This is because
 - $S \stackrel{*}{\Rightarrow} A_{q_0, q_a} w$ iff $\langle q_0, \epsilon \rangle \xrightarrow{P_N} \langle q_a, \epsilon \rangle$ (by previous proposition) iff $w \in L(P_N)$

Tying all the Ends

Proposition

Let P be a PDA then $L(P)$ is context-free.

Proof.

- A normalized PDA P_N can be constructed such that $L(P) = L(P_N)$
- A grammar G_P can be constructed such that $L(G_P) = L(P_N) = L(P)$. This is because
 - $S \stackrel{*}{\Rightarrow} A_{q_0, q_a} w$ iff $\langle q_0, \epsilon \rangle \xrightarrow{w}_{P_N} \langle q_a, \epsilon \rangle$ (by previous proposition) iff $w \in L(P_N) = L(P)$

