Languages and Machines

L7: CFGs and Pushdown Machines

Jorge A. Pérez

Bernoulli Institute for Math, Computer Science, and AI
University of Groningen, Groningen, the Netherlands

May 9, 2023

Languages and Their Machines

Regular	$\leftrightarrow$	Finite State Machines (FSMs)
Context-free	$\leftrightarrow$	**Pushdown Machines**
Context-sensitive	$\leftrightarrow$	Linearly-bounded Machines
Decidable	$\leftrightarrow$	Always-terminating Turing Machines
Semi-decidable	$\leftrightarrow$	Turing Machines

Outline

Context-Free Grammars

A context-free grammar is

$$G = (V, \Sigma, P, S)$$

where:

- V is a set of non-terminals
- Σ is a set of terminals
- P is a set of production rules
- S is the starting symbol

Balanced Parentheses

An archetypical example of a context-free language:
the set of balanced strings of parentheses ‘$[$’ and ‘$]$’.

A string of parenthesis is balanced if:

1. Each left parenthesis has a matching right parenthesis.
2. Matched pairs are well nested.

Balanced Parentheses

An archetypical example of a context-free language:
the set of balanced strings of parentheses '$[$' and '$]$'.

A string of parenthesis is balanced if:

1. Each left parenthesis has a matching right parenthesis.
2. Matched pairs are well nested.

For instance, '$[\,[\,]\,[\,]\,]$' is balanced but '$]\,[$' and '$[\,[\,]\,[\,[\,[\,]\,]$' are not.

It is generated by the following grammar:

$$S \to [\,S\,] \mid S\ S \mid \epsilon$$

Balanced Parentheses

Given a string of parentheses x, let us write $L(x)$ and $R(x)$ to denote the number of left and right parentheses in x.

Formally, a string of parentheses x is balanced if and only if

(i) $L(x) = R(x)$

(ii) for all prefixes y of x, $L(y) \geq R(y)$

Balanced Parentheses

Given a string of parentheses x, let us write $L(x)$ and $R(x)$ to denote the number of left and right parentheses in x.

Formally, a string of parentheses x is balanced if and only if

(i) $L(x) = R(x)$

(ii) for all prefixes y of x, $L(y) \geq R(y)$

Conditions (i) and (ii) are both necessary and sufficient for a formal definition of balanced parentheses.

Example: '$]$ $[$' satisfies (i) but not (ii).

Balanced Parentheses

Consider conditions (i) and (ii) in the previous slide. We have:

Theorem

Let G be the CFG

$$S \to [\, S\,] \mid S\, S \mid \epsilon$$

Then

$$L(G) = \{x \in \{[,]\}^* \mid x \textit{ satisfies conditions (i) and (ii)}\}$$

As usual, the proof proceeds by showing two directions:

1. If $S \Rightarrow^* x$ then x satisfies (i) and (ii)
2. If x is balanced then $S \Rightarrow^* x$

Direction 1: Proof Sketch

Induction on the length of the derivation $S \Rightarrow^*_G \alpha$, where α is a

Direction 1: Proof Sketch

Induction on the length of the derivation $S \Rightarrow^*_G \alpha$, where α is a sentential form (not necessarily a sentence)

Direction 1: Proof Sketch

Induction on the length of the derivation $S \Rightarrow_G^* \alpha$, where α is a sentential form (not necessarily a sentence)

- **Base case**: Immediate, for S trivially satisfies (i) and (ii)

Direction 1: Proof Sketch

Induction on the length of the derivation $S \Rightarrow_G^* \alpha$, where α is a sentential form (not necessarily a sentence)

- **Base case**: Immediate, for S trivially satisfies (i) and (ii)
- **Inductive case**: We focus on a sentential form β such that

$$S \Rightarrow^n \beta \Rightarrow \alpha$$

  By IH, β satisfies (i) and (ii). A case analysis on the production rule that could have been applied in the step from β to α:

Direction 1: Proof Sketch

Induction on the length of the derivation $S \Rightarrow_G^* \alpha$, where α is a sentential form (not necessarily a sentence)

- **Base case**: Immediate, for S trivially satisfies (i) and (ii)
- **Inductive case**: We focus on a sentential form β such that

$$S \Rightarrow^n \beta \Rightarrow \alpha$$

  By IH, β satisfies (i) and (ii). A case analysis on the production rule that could have been applied in the step from β to α:

  a. If $S \rightarrow \epsilon$ or $S \rightarrow S\,S$ was applied:
     Then the number/order of parentheses doesn't change, and the thesis holds easily

Direction 1: Proof Sketch

Induction on the length of the derivation $S \Rightarrow_G^* \alpha$, where α is a sentential form (not necessarily a sentence)

- **Base case**: Immediate, for S trivially satisfies (i) and (ii)
- **Inductive case**: We focus on a sentential form β such that

$$S \Rightarrow^n \beta \Rightarrow \alpha$$

  By IH, β satisfies (i) and (ii). A case analysis on the production rule that could have been applied in the step from β to α:

  a. If $S \rightarrow \epsilon$ or $S \rightarrow S\ S$ was applied:
     Then the number/order of parentheses doesn't change, and the thesis holds easily
  b. If $S \rightarrow [\ S\ ]$ was applied: This is the interesting case!

Direction 1: Proof Sketch

Induction on the length of the derivation $S \Rightarrow_G^* \alpha$, where α is a sentential form (not necessarily a sentence)

- **Base case**: Immediate, for S trivially satisfies (i) and (ii)
- **Inductive case**: We focus on a sentential form β such that

$$S \Rightarrow^n \beta \Rightarrow \alpha$$

  By IH, β satisfies (i) and (ii). A case analysis on the production rule that could have been applied in the step from β to α:

  a. If $S \to \epsilon$ or $S \to S\ S$ was applied:
  Then the number/order of parentheses doesn't change, and the thesis holds easily
  b. If $S \to [\ S\ ]$ was applied: This is the interesting case!
  Assume $\beta = \beta_1 S \beta_2$ and $\alpha = \beta_1 [S] \beta_2$.
  To show (i), we prove $L(\alpha) = R(\alpha)$, which follows from the IH.
  To show (ii), one checks prefixes γ of α. There are three cases: γ is a prefix of (a) β_1, (b) $\beta_1[S$, (c) $\beta_1[S]\delta$ (where δ is prefix of β_2).

Direction 2: Proof Sketch

To prove: If x is balanced (conditions (i) and (ii)) then $S \Rightarrow^* x$.
We apply induction on the length of x.

Direction 2: Proof Sketch

To prove: If x is balanced (conditions (i) and (ii)) then $S \Rightarrow^* x$.
We apply induction on the length of x.

- **Base case**:
  If $|x| = 0$ then $x = \epsilon$. The only possible production rule is $S \rightarrow \epsilon$.

Direction 2: Proof Sketch

To prove: If x is balanced (conditions (i) and (ii)) then $S \Rightarrow^* x$.
We apply induction on the length of x.

- **Base case**:
  If $|x| = 0$ then $x = \epsilon$. The only possible production rule is $S \to \epsilon$.
- **Inductive case**:
  We split the argument into two cases:
  a. There is a proper prefix y of x (i.e., $y \neq \epsilon$, $y \neq x$) that enjoys (i,ii)
  b. Such a proper prefix doesn't exist

  Intuition:
  If such a prefix y exists then we can deduce that we can derive x starting with the production $S \to S\ S$.
  Otherwise, x is of the form $[\ z\ ]$, for some z that enjoys (i,ii).
  We can derive x starting with the production $S \to [S]$.

Outline

Context-Free Grammars

Simple Pushdown Machines
- Examples

Variants of PDMs

CFGs and PDMs
- From CFGs to PDMs
- From PDMs to CFGs

Closure Properties for CFLs

Simple Pushdown Machines

A **pushdown machine** is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set (of states)
- Σ is the input alphabet
- Γ is the alphabet for the **stack**, a last in / first out structure.
- q_0 is a start state
- $F \subseteq Q$ is a set of accepting/final states
- δ is the transition function:

$$\delta : \; Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q \times (\Gamma \cup \{\epsilon\}))$$

Simple Pushdown Machines

A **pushdown machine** is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set (of states)
- Σ is the input alphabet
- Γ is the alphabet for the **stack**, a last in / first out structure.
- q_0 is a start state
- $F \subseteq Q$ is a set of accepting/final states
- δ is the transition function:

$$\delta : \underbrace{Q}_{\text{state}} \times \overbrace{(\Sigma \cup \{\epsilon\})}^{\text{input symbol}} \times \underbrace{(\Gamma \cup \{\epsilon\})}_{\text{symbol to pop off}} \rightarrow \mathcal{P}(\overbrace{Q}^{\text{new state}} \times \underbrace{(\Gamma \cup \{\epsilon\})}_{\text{symbol to push}})$$

Intuition:

For every triple (q, a, X), δ defines a set of pairs (r, Y)

- In state q, symbol a can be read **if** X is at the top of the stack
- A transition replaces X with Y, and the machine moves to r

Simple Pushdown Machines

A **pushdown machine** is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set (of states)
- Σ is the input alphabet
- Γ is the alphabet for the **stack**, a last in / first out structure.
- q_0 is a start state
- $F \subseteq Q$ is a set of accepting/final states
- δ is the transition function:

$$\delta : \; Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \to \mathcal{P}(Q \times (\Gamma \cup \{\epsilon\}))$$

Intuition:
For every triple (q, a, X), δ defines a set of pairs (r, Y)

- In state q, symbol a can be read **if** X is at the top of the stack
- A transition replaces X with Y, and the machine moves to r

Acceptance:
Scan full input, halt with empty stack **and** in a final state.

Example 1

We have $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{A\}$
- $F = \{q_1\}$
- The transition function δ:

$\delta(q_0, a, \epsilon) = \{(q_0, A)\}$	Add an A to the stack
$\delta(q_0, \epsilon, \epsilon) = \{(q_1, \epsilon)\}$	Non-deterministically move to q_1
$\delta(q_1, b, A) = \{(q_1, \epsilon)\}$	Remove an A from the stack

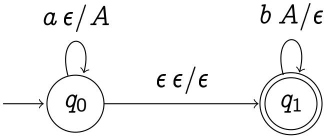
Example 1

We have $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{A\}$
- $F = \{q_1\}$
- The transition function δ:

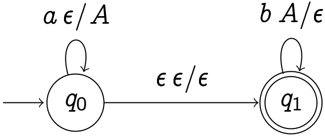
$$\begin{aligned}
\delta(q_0, a, \epsilon) &= \{(q_0, A)\} && \text{Add an } A \text{ to the stack} \\
\delta(q_0, \epsilon, \epsilon) &= \{(q_1, \epsilon)\} && \text{Non-deterministically move to } q_1 \\
\delta(q_1, b, A) &= \{(q_1, \epsilon)\} && \text{Remove an } A \text{ from the stack}
\end{aligned}$$

More conveniently:



Example 1 (continued)

Accepting $L_1 = \{a^n b^n \mid n \geq 0\}$:



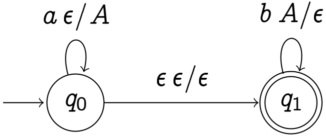
Key idea: Use stack symbol A to encode n.

Example:

- Input: $aaabbb$
- Stack: $[\epsilon\rangle$

Example 1 (continued)

Accepting $L_1 = \{a^n b^n \mid n \geq 0\}$:



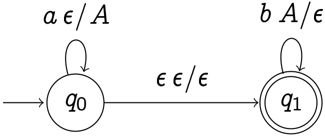
Key idea: Use stack symbol A to encode n.

Example:

- Input: $a \| aabbb$
- Stack: $[A\rangle$

Example 1 (continued)

Accepting $L_1 = \{a^n b^n \mid n \geq 0\}$:



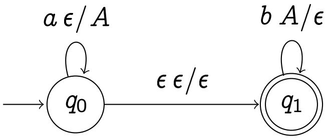
Key idea: Use stack symbol A to encode n.

Example:

- Input: $aa \,\|\, abbb$
- Stack: $[AA\rangle$

Example 1 (continued)

Accepting $L_1 = \{a^n b^n \mid n \geq 0\}$:



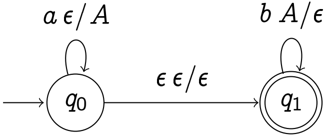
Key idea: Use stack symbol A to encode n.

Example:

- Input: $aaa \,\|\, bbb$
- Stack: $[AAA\rangle$

Example 1 (continued)

Accepting $L_1 = \{a^n b^n \mid n \geq 0\}$:



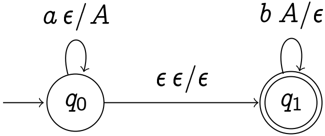
Key idea: Use stack symbol A to encode n.

Example:

- Input: $aaab \parallel bb$
- Stack: $[AA\rangle$

Example 1 (continued)

Accepting $L_1 = \{a^n b^n \mid n \geq 0\}$:



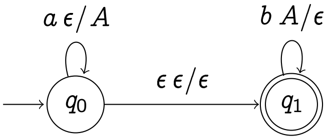
Key idea: Use stack symbol A to encode n.

Example:

- Input: $aaabb \,\|\, b$
- Stack: $[A\rangle$

Example 1 (continued)

Accepting $L_1 = \{a^n b^n \mid n \geq 0\}$:



Key idea: Use stack symbol A to encode n.

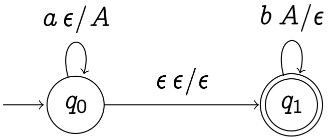
Example:

- Input: $aaabbb\|$
- Stack: $[\epsilon\rangle$

✓ No input to read, empty stack, q_1 is accepting

Example 1 (continued)

Accepting $L_1 = \{a^n b^n \mid n \geq 0\}$:



Key idea: Use stack symbol A to encode n.

Example:

- Input: $aaabbb||$
- Stack: $[\epsilon\rangle$

✓ No input to read, empty stack, q_1 is accepting

In contrast:

✗ abb is not accepted: symbol b is left over, with an empty stack

✗ aab is not accepted: no symbols to read, the stack is not empty

Example 2

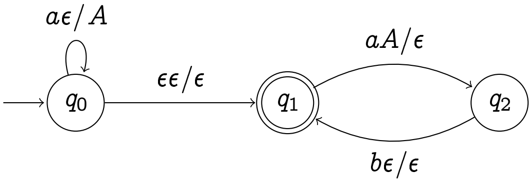
Construct a simple PDM that accepts the language:

$$L_2 = \{a^i(ab)^i \mid i \geq 0\}$$

Example 2

Construct a simple PDM that accepts the language:

$$L_2 = \{a^i(ab)^i \mid i \geq 0\}$$



Configurations and Acceptance

A **configuration** for a PDM is defined as a triple $[q, w, \beta]$ with

- $q \in Q$: the current state
- $w \in \Sigma^*$: the remainder of the input
- $\beta \in \Gamma^*$: the current contents of the stack

The transition relation $\vdash$ indicates the steps that the PDM can take:

$$[q, aw, X\gamma] \vdash [r, w, Y\gamma] \equiv (r, Y) \in \delta(q, a, X)$$

Intuitively:
If
in state q symbol a is read from the input, symbol X is popped from the stack, and $(r, Y) \in \delta(q, a, X)$
then
the PDM can push symbol Y onto the stack and move to state r.

Configurations and Acceptance

A **configuration** for a PDM is defined as a triple $[q, w, \beta]$ with

- $q \in Q$: the current state
- $w \in \Sigma^*$: the remainder of the input
- $\beta \in \Gamma^*$: the current contents of the stack

The transition relation $\vdash$ indicates the steps that the PDM can take:

$$[q, aw, X\gamma] \vdash [r, w, Y\gamma] \equiv (r, Y) \in \delta(q, a, X)$$

The language accepted by a PDM:

$$L(M) = \{w \in \Sigma^* \mid \exists q \in F : [q_0, w, \epsilon] \vdash^* [q, \epsilon, \epsilon]\}$$

Acceptance by **accepting state** and **empty stack.**

Outline

Variants of PDMs

Variations on the machine itself:

- Atomic PDMs
  Each transition performs one of three actions:
  pop the stack, push onto the stack, process an input symbol
- Extended PDMs
  Transitions push strings of symbols onto the stack, rather than just one symbol

Variations on acceptance:

- By accepting state only (the stack may be not empty)
- By empty stack only (final state may not be accepting)

All variants are equivalent to simple PDMs (with acceptance by both accepting state and empty stack)

Atomic and Extended PDMs

Atomic PDMs:

- Transitions have the form:

$$
\begin{array}{rl}
(q_j, \epsilon) \in \delta(q_i, a, \epsilon) & \text{[read an input symbol]} \\
(q_j, \epsilon) \in \delta(q_i, \epsilon, A) & \text{[pop a stack element]} \\
(q_j, A) \in \delta(q_i, \epsilon, \epsilon) & \text{[push a stack element]}
\end{array}
$$

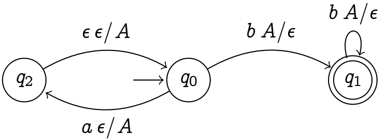
Extended PDMs:

- Push a sequence of symbols onto the stack at the same time
- We modify the transition relation: from $Q \times \Gamma$ to $Q \times \Gamma^*$:

$$\delta : \; Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \to \mathcal{P}(Q \times \Gamma^*)$$

Comparison

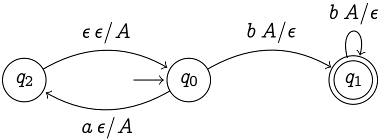
PDM:



Q: What is the language recognized?

Comparison

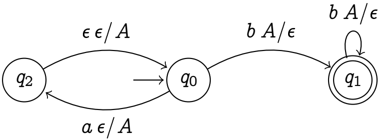
PDM:



Q: What is the language recognized? A: $\{a^i\ b^{2i} \mid i \geq 1\}$

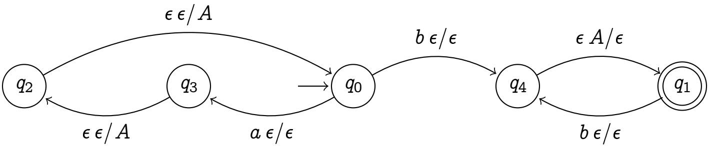
Comparison

PDM:



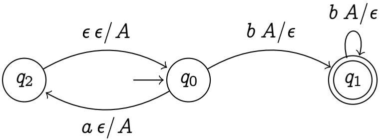
Q: What is the language recognized? A: $\{a^i\, b^{2i} \mid i \geq 1\}$

Atomic PDM:



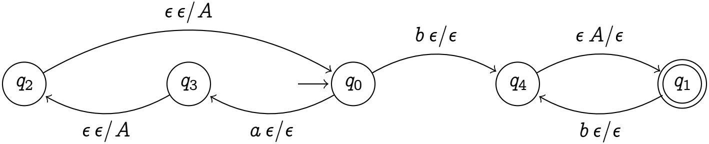
Comparison

PDM:

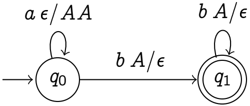


Q: What is the language recognized? A: $\{a^i\, b^{2i} \mid i \geq 1\}$

Atomic PDM:

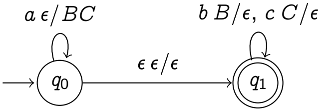


Extended PDM:



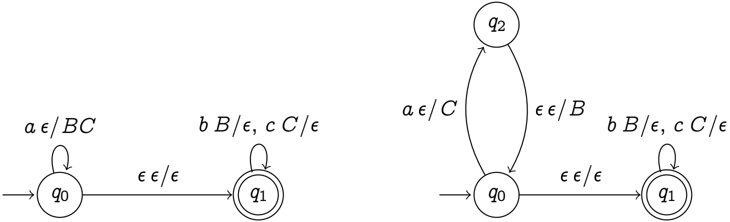
Extended PDMs

Example: An extended PDM, and its corresponding simple PDM



Extended PDMs

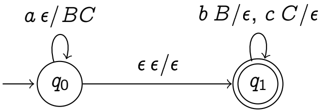
Example: An extended PDM, and its corresponding simple PDM

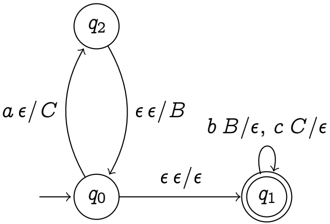


Q: What is the language recognized?

Extended PDMs

Example: An extended PDM, and its corresponding simple PDM

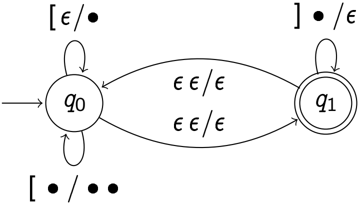




Q: What is the language recognized? A: $\{a^n(bc)^n \mid n \geq 0\}$

Example

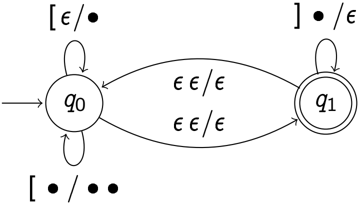
Recognizing balanced parentheses:



- If [is read, and the stack is empty: push • onto the stack
- If] is read, and the stack has •: push • • onto the stack
- If] is read, and the stack has •: remove the • from the stack

Example

Recognizing balanced parentheses:



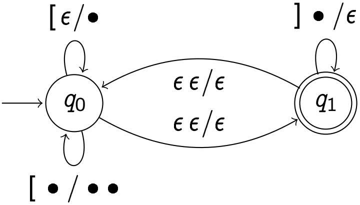
- If [is read, and the stack is empty: push • onto the stack
- If] is read, and the stack has •: push • • onto the stack
- If] is read, and the stack has •: remove the • from the stack

The • in the stack represent open (left) parentheses. Example:

- Input: || [[ ] [ ]]
- Stack: ϵ

Example

Recognizing balanced parentheses:



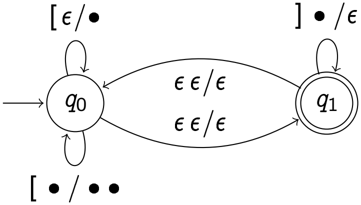
- If [is read, and the stack is empty: push • onto the stack
- If] is read, and the stack has •: push • • onto the stack
- If] is read, and the stack has •: remove the • from the stack

The • in the stack represent open (left) parentheses. Example:

- Input: [‖ [ ] [ ]]
- Stack: •

Example

Recognizing balanced parentheses:



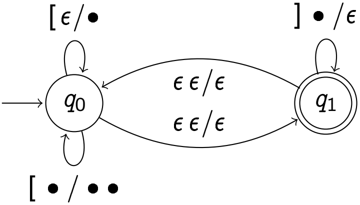
- If [is read, and the stack is empty: push • onto the stack
- If] is read, and the stack has •: push • • onto the stack
- If] is read, and the stack has •: remove the • from the stack

The • in the stack represent open (left) parentheses. Example:

- Input: [[‖] [ ]]
- Stack: ••

Example

Recognizing balanced parentheses:



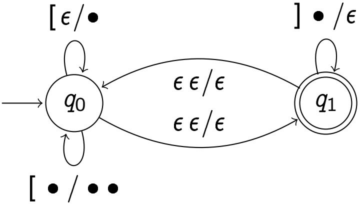
- If [is read, and the stack is empty: push • onto the stack
- If] is read, and the stack has •: push • • onto the stack
- If] is read, and the stack has •: remove the • from the stack

The • in the stack represent open (left) parentheses. Example:

- Input: [[ ] ‖ [ ]]
- Stack: •

Example

Recognizing balanced parentheses:



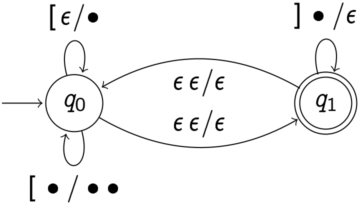
- If [is read, and the stack is empty: push • onto the stack
- If] is read, and the stack has •: push • • onto the stack
- If] is read, and the stack has •: remove the • from the stack

The • in the stack represent open (left) parentheses. Example:

- Input: [[ ] [ ]]
- Stack: ••

Example

Recognizing balanced parentheses:



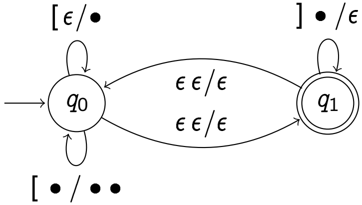
- If [is read, and the stack is empty: push • onto the stack
- If] is read, and the stack has •: push • • onto the stack
- If] is read, and the stack has •: remove the • from the stack

The • in the stack represent open (left) parentheses. Example:

- Input: [[ ] [ ] ||]
- Stack: •

Example

Recognizing balanced parentheses:



- If [is read, and the stack is empty: push • onto the stack
- If] is read, and the stack has •: push • • onto the stack
- If] is read, and the stack has •: remove the • from the stack

The • in the stack represent open (left) parentheses. Example:

- Input: [[ ] [ ]]
- Stack: ϵ

Variants of PDMs

We have seen: acceptance by **accepting state** and **empty stack**:

$$L(M) = \{w \in \Sigma^* \mid \exists q \in F : [q_0, w, \epsilon] \vdash^* [q, \epsilon, \epsilon]\}$$

Variations on acceptance:

1. By accepting state only (the stack may be not empty):

$$L(M) = \{w \in \Sigma^* \mid \exists q \in F, \alpha \in \Gamma^* : [q_0, w, \epsilon] \vdash^* [q, \epsilon, \alpha]\}$$

2. By empty stack only (final state may not be accepting)

$$L(M) = \{w \in \Sigma^* \mid \exists q \in Q : [q_0, w, \epsilon] \vdash^* [q, \epsilon, \epsilon]\}$$

Variants of PDMs

We have seen: acceptance by **accepting state** and **empty stack**:

$$L(M) = \{w \in \Sigma^* \mid \exists q \in F : [q_0, w, \epsilon] \vdash^* [q, \epsilon, \epsilon]\}$$

Variations on acceptance:

1. By accepting state only (the stack may be not empty):

$$L(M) = \{w \in \Sigma^* \mid \exists q \in F, \alpha \in \Gamma^* : [q_0, w, \epsilon] \vdash^* [q, \epsilon, \alpha]\}$$

2. By empty stack only (final state may not be accepting)

$$L(M) = \{w \in \Sigma^* \mid \exists q \in Q : [q_0, w, \epsilon] \vdash^* [q, \epsilon, \epsilon]\}$$

All variants are equivalent to simple PDMs (with acceptance by both accepting state and empty stack):

1. Give a machine M' with new transitions that empty the stack.
2. Give an M' identical to M, with all states defined as accepting.

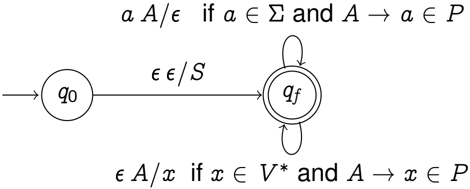
Outline

From Context-Free Grammars to PDMs

- Extended PDMs may represent grammars $G = (V, \Sigma, P, S)$
- Assume G is normalized: for every $A \to w \in P$, w is either a single terminal (in Σ) or a string of non-terminals (in V^*)

From Context-Free Grammars to PDMs

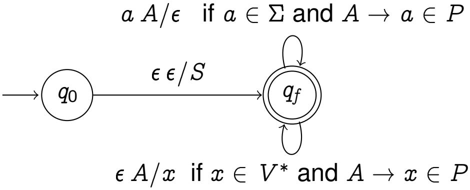
- Extended PDMs may represent grammars $G = (V, \Sigma, P, S)$
- Assume G is normalized: for every $A \to w \in P$, w is either a single terminal (in Σ) or a string of non-terminals (in V^*)
- We construct a PDM M such that $L(M) = L(G)$:



- Notice: the stack only stores non-terminals in V

From Context-Free Grammars to PDMs

- Extended PDMs may represent grammars $G = (V, \Sigma, P, S)$
- Assume G is normalized: for every $A \to w \in P$, w is either a single terminal (in Σ) or a string of non-terminals (in V^*)
- We construct a PDM M such that $L(M) = L(G)$:



- Notice: the stack only stores non-terminals in V
- Given $w \in \Sigma^*$, we have the following equivalence:

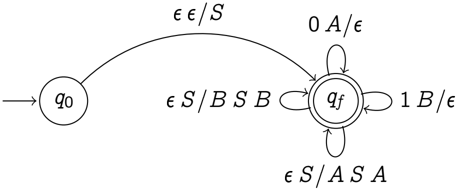
$$[q_f, w, S] \vdash^* [q_f, v, \alpha] \equiv \exists u \in \Sigma^* : w = uv \land S \Rightarrow^*_{lm} u\alpha$$

From CFGs to PDMs: Example

Given the normalized grammar

$$S \to A\ S\ A \mid B\ S\ B \mid \epsilon \qquad A \to 0 \qquad B \to 1$$

We have the following extended PDM:

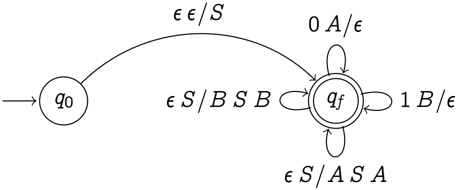


From CFGs to PDMs: Example

Given the normalized grammar

$$S \to A\ S\ A \mid B\ S\ B \mid \epsilon \qquad A \to 0 \qquad B \to 1$$

We have the following extended PDM:

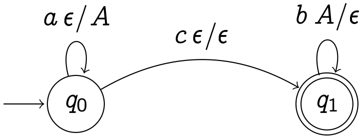


We can check:

$$[q_0, 1001, \epsilon] \vdash [q_1, 1001, S] \vdash [q_1, 1001, B\ S\ B] \vdash$$
$$[q_1, 001, S\ B] \vdash [q_1, 001, A\ S\ A\ B] \vdash [q_1, 01, S\ A\ B] \vdash$$
$$[q_1, 01, A\ B] \vdash [q_1, 1, B] \vdash [q_1, \epsilon, \epsilon]$$

From PDMs to CFGs

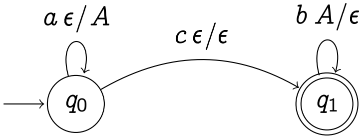
Consider the simple PDM M:



Q: What is $L(M)$?

From PDMs to CFGs

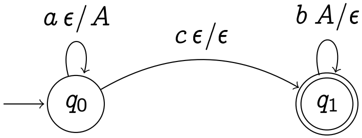
Consider the simple PDM M:



Q: What is $L(M)$? A: $\{a^n cb^n \mid n \geq 0\}$.

From PDMs to CFGs

Consider the simple PDM M:



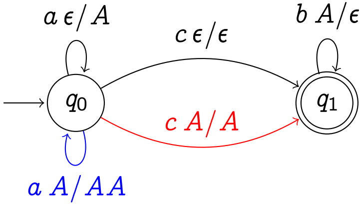
Q: What is $L(M)$? A: $\{a^n cb^n \mid n \geq 0\}$.

A recipe to show that $L(M)$ is context-free:

1. Convert M into an extended PDM M' by augmenting transitions
2. Use the transitions of M' to construct the rules P in $G = (V, \Sigma, P, S)$
   Key idea: Use as non-terminals objects of the form $\langle q_i, A, q_j \rangle$, where q_i, q_j are states of M', and $A \in \Gamma \cup \{\epsilon\}$

From PDMs to CFGs: Step 1

Construct the extended PDM M' (with transition function δ'):

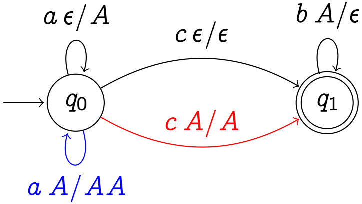


We look at the transitions in M that don't remove elements from the stack, and add new transitions to M' accordingly, using Γ:

- If $(q_j, B) \in \delta(q_i, u, \epsilon)$, then $\delta'(q_i, u, A) = \{(q_j, BA) \mid A \in \Gamma\}$
- If $(q_j, \epsilon) \in \delta(q_i, u, \epsilon)$, then $\delta'(q_i, u, A) = \{(q_j, A) \mid A \in \Gamma\}$

From PDMs to CFGs: Step 1

Construct the extended PDM M' (with transition function δ'):



We look at the transitions in M that don't remove elements from the stack, and add new transitions to M' accordingly, using Γ:

- If $(q_j, B) \in \delta(q_i, u, \epsilon)$, then $\delta'(q_i, u, A) = \{(q_j, BA) \mid A \in \Gamma\}$
- If $(q_j, \epsilon) \in \delta(q_i, u, \epsilon)$, then $\delta'(q_i, u, A) = \{(q_j, A) \mid A \in \Gamma\}$

New transitions: $\delta(q_0, a, A) = \{(q_0, AA)\}$ and $\delta(q_0, c, A) = \{(q_1, A)\}$.

From PDMs to CFGs: Step 2

Use M' to construct the grammar $G = (V, \Sigma, P, S)$ as follows:

- Σ is the input alphabet of M'
- V consists of a start symbol S and objects of the form $\langle q_i, A, q_j \rangle$, where q_i, q_j are states of M', and $A \in \Gamma \cup \{\epsilon\}$.
- $\langle q_i, A, q_j \rangle$: a run that begins in q_i, ends in q_j, and removes A.

From PDMs to CFGs: Step 2

Use M' to construct the grammar $G = (V, \Sigma, P, S)$ as follows:

- Σ is the input alphabet of M'
- V consists of a start symbol S and objects of the form $\langle q_i, A, q_j \rangle$, where q_i, q_j are states of M', and $A \in \Gamma \cup \{\epsilon\}$.
- $\langle q_i, A, q_j \rangle$: a run that begins in q_i, ends in q_j, and removes A.

The production rules in P are constructed as follows:

I. $S \rightarrow \langle q_0, \epsilon, q_j \rangle$, for each $q_j \in F$.

II. Each $(q_j, B) \in \delta(q_i, x, A)$ (with $A \in \Gamma \cup \{\epsilon\}$), generates the set:

$$\{\langle q_i, A, q_k \rangle \rightarrow x \; \langle q_j, B, q_k \rangle \mid q_k \in Q\}$$

III. Each $(q_j, BA) \in \delta(q_i, x, A)$ (with $A \in \Gamma$), generates the set:

$$\{\langle q_i, A, q_k \rangle \rightarrow x \; \langle q_j, B, q_n \rangle \; \langle q_n, A, q_k \rangle \mid q_n, q_k \in Q\}$$

IV. For each $q_k \in Q$, we have $\langle q_k, \epsilon, q_k \rangle \rightarrow \epsilon$

From Transitions to Rules

–	1	$S \to \langle q_0, \epsilon, q_1 \rangle$
$\delta(q_0, a, \epsilon) = \{(q_0, A)\}$	2	$\langle q_0, \epsilon, q_0 \rangle \to a \langle q_0, A, q_0 \rangle$
	3	$\langle q_0, \epsilon, q_1 \rangle \to a \langle q_0, A, q_1 \rangle$
$\delta(q_0, a, A) = \{(q_0, AA)\}$	4	$\langle q_0, A, q_0 \rangle \to a \langle q_0, A, q_0 \rangle \langle q_0, A, q_0 \rangle$
	5	$\langle q_0, A, q_1 \rangle \to a \langle q_0, A, q_0 \rangle \langle q_0, A, q_1 \rangle$
	6	$\langle q_0, A, q_0 \rangle \to a \langle q_0, A, q_1 \rangle \langle q_1, A, q_0 \rangle$
	7	$\langle q_0, A, q_1 \rangle \to a \langle q_0, A, q_1 \rangle \langle q_1, A, q_1 \rangle$
$\delta(q_0, c, \epsilon) = \{(q_1, \epsilon)\}$	8	$\langle q_0, \epsilon, q_0 \rangle \to c \langle q_1, \epsilon, q_0 \rangle$
	9	$\langle q_0, \epsilon, q_1 \rangle \to c \langle q_1, \epsilon, q_1 \rangle$
$\delta(q_0, c, A) = \{(q_1, A)\}$	10	$\langle q_0, A, q_0 \rangle \to c \langle q_1, A, q_0 \rangle$
	11	$\langle q_0, A, q_1 \rangle \to c \langle q_1, A, q_1 \rangle$
$\delta(q_1, b, A) = \{(q_1, \epsilon)\}$	12	$\langle q_1, A, q_0 \rangle \to b \langle q_1, \epsilon, q_0 \rangle$
	13	$\langle q_1, A, q_1 \rangle \to b \langle q_1, \epsilon, q_1 \rangle$
–	14	$\langle q_0, \epsilon, q_0 \rangle \to \epsilon$
	15	$\langle q_1, \epsilon, q_1 \rangle \to \epsilon$

Example

We can check that the sequence of transitions

$$\begin{aligned}[q_0, aacbb, \epsilon] &\vdash [q_0, acbb, A] \\ &\vdash [q_0, cbb, AA] \\ &\vdash [q_1, bb, AA] \\ &\vdash [q_1, b, A] \\ &\vdash [q_1, \epsilon, \epsilon]\end{aligned}$$

is mimicked by Rules 1, 3, 7, 11, 13, 15, 13, 15 in the previous slide:

$$\begin{aligned}S &\Rightarrow_1 \langle q_0, \epsilon, q_1\rangle \Rightarrow_3 a\underline{\langle q_0, A, q_1\rangle} \\ &\Rightarrow_7 aa\underline{\langle q_0, A, q_1\rangle}\langle q_1, A, q_1\rangle \\ &\Rightarrow_{11} aac\underline{\langle q_1, A, q_1\rangle}\langle q_1, A, q_1\rangle \\ &\Rightarrow_{13} aacb\underline{\langle q_1, \epsilon, q_1\rangle}\langle q_1, A, q_1\rangle \\ &\Rightarrow_{15} aacb\underline{\langle q_1, A, q_1\rangle} \Rightarrow_{13} aacbb\underline{\langle q_1, \epsilon, q_1\rangle} \Rightarrow_{15} aacbb\end{aligned}$$

Outline

Closure Properties for CFLs

- CFLs are closed under union, concatenation, and Kleene star
  In all cases: construct a CFG from the CFGs of the given CFLs

Closure Properties for CFLs

- CFLs are closed under union, concatenation, and Kleene star
  In all cases: construct a CFG from the CFGs of the given CFLs
- CFLs are *not* closed under intersection
  Take CFLs $L_1 = \{a^i b^i c^k \mid i, k \in \mathbb{N}\}$, $L_2 = \{a^i b^k c^k \mid i, k \in \mathbb{N}\}$.
  But $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ is not a CFL
  (cf. Pumping Lemma for CFLs).
- CFLs are *not* closed under complementation
  Assume, for a contradiction, closure under complementation.
  Let L_1, L_2 be any CFLs. Then $L = \overline{\overline{L_1} \cup \overline{L_2}}$ is CFL.
  Now, by De Morgan's law, $L = L_1 \cap L_2$; this contradicts the above.

Closure Properties for CFLs

- CFLs are closed under union, concatenation, and Kleene star
  In all cases: construct a CFG from the CFGs of the given CFLs

- CFLs are *not* closed under intersection
  Take CFLs $L_1 = \{a^i b^i c^k \mid i, k \in \mathbb{N}\}$, $L_2 = \{a^i b^k c^k \mid i, k \in \mathbb{N}\}$.
  But $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ is not a CFL
  (cf. Pumping Lemma for CFLs).

- CFLs are *not* closed under complementation
  Assume, for a contradiction, closure under complementation.
  Let L_1, L_2 be any CFLs. Then $L = \overline{\overline{L_1} \cup \overline{L_2}}$ is CFL.
  Now, by De Morgan's law, $L = L_1 \cap L_2$; this contradicts the above.

- If R is a regular language and L is a CFL, then $R \cap L$ is CFL
  Take a DFSM recognizing R and a simple PDM recognizing L.
  Build a PDM that applies both machines simultaneously.

Taking Stock

- Context-free languages/grammars
- Balanced parenthesis
- Pushdown machines (PDMs): simple and extended
- From CFGs to PDMs
- From PDMs to CFGs
- Closure properties

We didn't cover (self study!):

- Pumping Lemma for CFLs (Sect 4.2)

Next Lecture(s)

- Turing machines