Languages and Machines

L7: CFLs and Pushdown Machines

Dan Frumin

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

May 7, 2024

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 (e.g. $A \rightarrow abAb$)
- $S \in V$ 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.

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:

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

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

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

Then

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

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 $S \Rightarrow_G^* \alpha$.

Direction 1: Proof Sketch

Induction on the length of the derivation $S \Rightarrow_G^* \alpha$.

But in general, α is an arbitrary sentential form (not necessary a word). We say that α is balanced if the string that we obtain from α by erasing all the non-terminals is balanced.
E.g. $[[S]]$ is balanced, $[]S[$ is not.

Direction 1: Proof Sketch

Induction on the length of the derivation $S \Rightarrow_G^* \alpha$.

But in general, α is an arbitrary sentential form (not necessary a word). We say that α is balanced if the string that we obtain from α by erasing all the non-terminals is balanced.
E.g. $[[S]]$ is balanced, $[]S[$ is not.

Base case. The "empty" derivation $S \Rightarrow_G^* S$. Erasing non-terminals we get an empty string, which is balanced.

Direction 1: Proof Sketch

Inductive case. We focus on a sentential form β such that

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

By IH, β is balanced.

Direction 1: Proof Sketch

Inductive case. We focus on a sentential form β such that

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

By IH, β is balanced.
A case analysis on the production rule that could have been applied in the step from β to α:

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

Direction 1: Proof Sketch

Inductive case. We focus on a sentential form β such that

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

By IH, β is balanced.
A case analysis on the production rule that could have been applied in the step from β to α:

a. If $S \rightarrow \varepsilon$ 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

Inductive case. We focus on a sentential form β such that

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

By IH, β is balanced.
A case analysis on the production rule that could have been applied in the step from β to α:

a. If $S \to \varepsilon$ 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!

   $$S \Rightarrow_G^n \beta_1 S \beta_2 \Rightarrow_G \beta_1 [S] \beta_2$$

   Condition (i) 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 $x = \varepsilon$. The only possible production rule is $S \to \varepsilon$.

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 $x = \varepsilon$. The only possible production rule is $S \to \varepsilon$.
- **Inductive case**:
  We split the argument into two cases:
  a. There is a proper prefix y of x (i.e., $y \neq \varepsilon, y \neq x$) 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

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

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

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

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

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

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

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:

$$\delta(q_0, a, \varepsilon) = \{(q_0, A)\} \qquad \text{Add an } A \text{ to the stack}$$

$$\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \varepsilon)\} \qquad \text{Non-deterministically move to } q_1$$

$$\delta(q_1, b, A) = \{(q_1, \varepsilon)\} \qquad \text{Remove an } A \text{ from the stack}$$

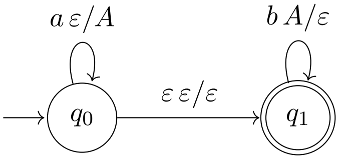
Example 1

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:

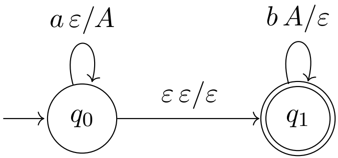
$\delta(q_0, a, \varepsilon) = \{(q_0, A)\}$	Add an A to the stack
$\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \varepsilon)\}$	Non-deterministically move to q_1
$\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$	Remove an A from the stack

More conveniently:



Example 1 (continued)

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



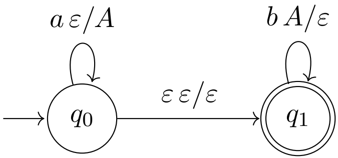
Key idea: Use stack symbol A to encode n.

Example:

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

Example 1 (continued)

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



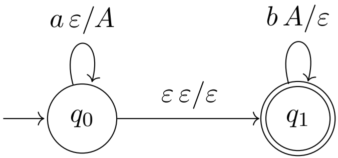
Key idea: Use stack symbol A to encode n.

Example:

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

Example 1 (continued)

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



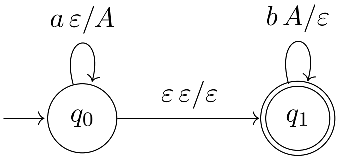
Key idea: Use stack symbol A to encode n.

Example:

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

Example 1 (continued)

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



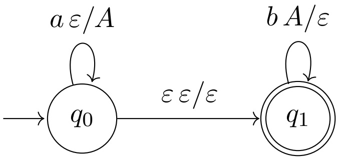
Key idea: Use stack symbol A to encode n.

Example:

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

Example 1 (continued)

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



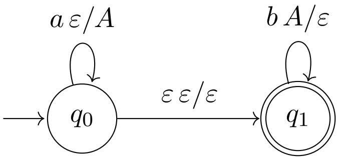
Key idea: Use stack symbol A to encode n.

Example:

- Input: $aaab \,{\color{red}\|}\, bb$
- Stack: $[AA\rangle$

Example 1 (continued)

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



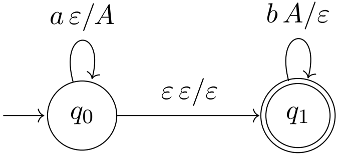
Key idea: Use stack symbol A to encode n.

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.

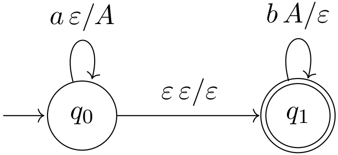
Example:

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

✓ 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: $[\varepsilon\rangle$

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

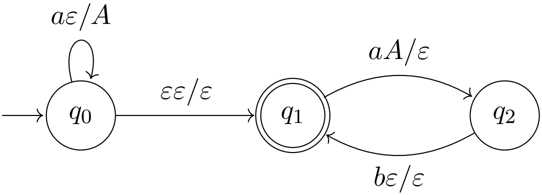
In contrast:

✗ 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

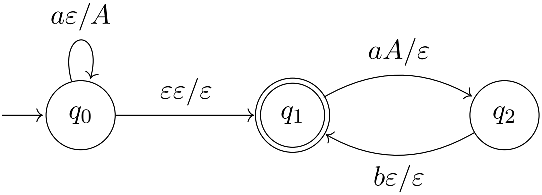
Example 2

A simple PDM that accepts the language L_2:



Example 2

A simple PDM that accepts the language L_2:



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

Configurations and Acceptance

We want to define a *step* relation $\vdash_M$ between configurations.

Configurations and Acceptance

We want to define a *step* relation $\vdash_M$ between configurations.
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

Then:

- $[q, aw, X\gamma] \vdash_M [r, w, Y\gamma]$ if $(r, Y) \in \delta(q, a, X)$
- $[q, w, X\gamma] \vdash_M [r, w, Y\gamma]$ if $(r, Y) \in \delta(q, \varepsilon, X)$

Both X and Y can be ε.

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

We want to define a *step* relation $\vdash_M$ between configurations.
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

Then:

- $[q, aw, X\gamma] \vdash_M [r, w, Y\gamma]$ if $(r, Y) \in \delta(q, a, X)$
- $[q, w, X\gamma] \vdash_M [r, w, Y\gamma]$ if $(r, Y) \in \delta(q, \varepsilon, X)$

Both X and Y can be ε.

The language accepted by a PDM:

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

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

Outline

Variants of PDMs

Variations on acceptance:

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

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

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

Variations on acceptance

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

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

Alternatives

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

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

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

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

Variations on acceptance

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

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

Alternatives

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

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

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

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

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

Variations on acceptance

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

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

Variations on acceptance

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

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

Give a machine M' with new transitions that empty the stack.

Variations on acceptance

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

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

Variations on acceptance

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

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

Give an M' identical to M, with all states defined as accepting.

Atomic and Extended PDMs

Atomic PDMs:

- Transitions have the form:

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

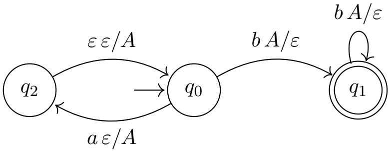
Extended PDMs:

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

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

Comparison

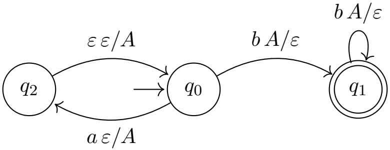
Simple PDM:



Q: What is the language recognized?

Comparison

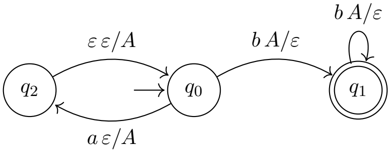
Simple PDM:



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

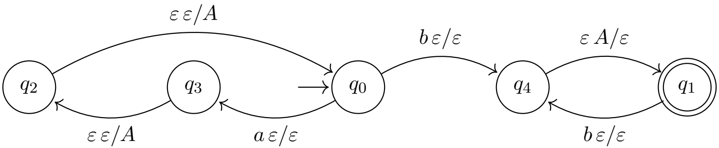
Comparison

Simple PDM:



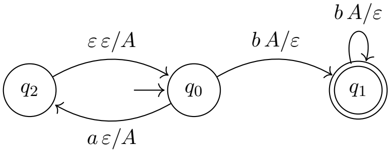
Q: What is the language recognized? A: $\{a^i\, b^{2i} \mid i \geq 1\}$

Atomic PDM:



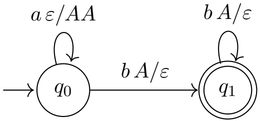
Comparison

Simple PDM:



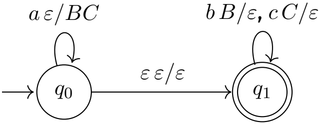
Q: What is the language recognized? A: $\{a^i\, b^{2i} \mid i \geq 1\}$

Extended PDM:



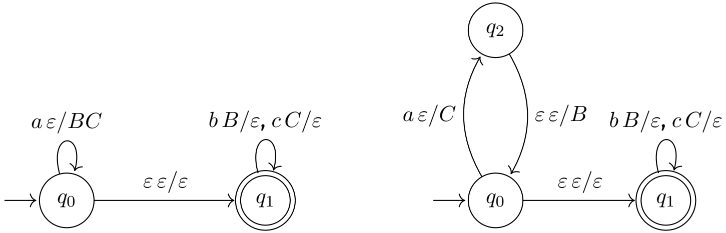
Extended PDMs

Example: An extended PDM, and its corresponding simple PDM



Extended PDMs

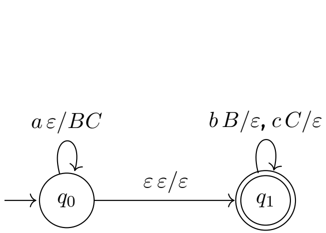
Example: An extended PDM, and its corresponding simple PDM

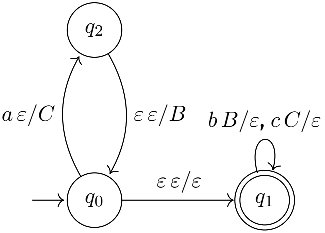


Q: What is the language recognized?

Extended PDMs

Example: An extended PDM, and its corresponding simple PDM





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

Outline

CFGs and call stacks

Production rules like $A \to a_1 A_1 \ldots A_n$ can be read "operationally" as a procedure A which:

1. read a_1
2. call procedure A_1
3. $\ldots$
4. call procedure A_n

This corresponds to a *leftmost* derivation

CFGs and call stacks

Production rules like $A \to a_1 A_1 \ldots A_n$ can be read “operationally” as a procedure A which:

1. read a_1
2. call procedure A_1
3. $\ldots$
4. call procedure A_n

This corresponds to a *leftmost* derivation

In regular grammars we only have $A \to aB$ which “operationally” corresponds to:

1. read a
2. call B

CFGs and call stacks

Production rules like $A \to a_1 A_1 \ldots A_n$ can be read "operationally" as a procedure A which:

1. read a_1
2. call procedure A_1
3. $\ldots$
4. call procedure A_n

This corresponds to a *leftmost* derivation

In regular grammars we only have $A \to aB$ which "operationally" corresponds to:

1. read a
2. call B

What do we need to implement these patterns?

CFGs and call stacks

Production rules like $A \to a_1 A_1 \ldots A_n$ can be read "operationally" as a procedure A which:

1. read a_1
2. call procedure A_1
3. $\ldots$
4. call procedure A_n

This corresponds to a *leftmost* derivation

In regular grammars we only have $A \to aB$ which "operationally" corresponds to:

1. read a
2. call B

What do we need to implement these patterns? Call stack for CFGs, just jumps to RGs.

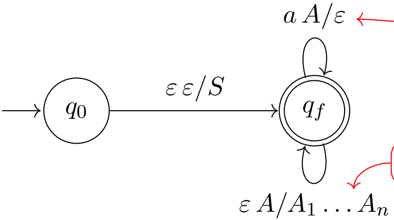
From Context-Free Grammars to PDMs

- Assume we have G a normalized CFG: for every $A \to w \in P$, w is either a single terminal or a string of non-terminals

$$A \to a \qquad\qquad A \to BCD$$

From Context-Free Grammars to PDMs

- Assume we have G a normalized CFG: for every $A \to w \in P$, w is either a single terminal or a string of non-terminals
- We construct a PDM M such that $L(M) = L(G)$:

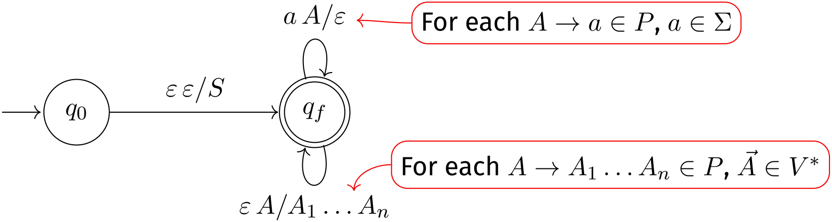


For each $A \to a \in P$, $a \in \Sigma$

For each $A \to A_1 \ldots A_n \in P$, $\vec{A} \in V^*$

From Context-Free Grammars to PDMs

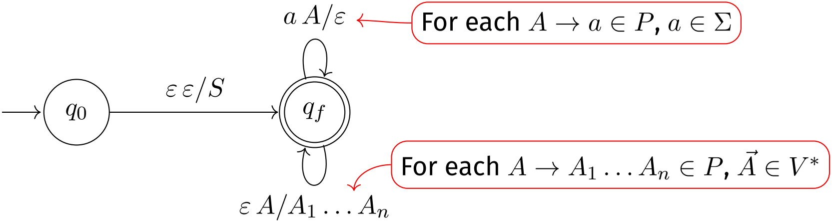
- Assume we have G a normalized CFG: for every $A \to w \in P$, w is either a single terminal or a string of non-terminals
- We construct a PDM M such that $L(M) = L(G)$:



- Notice: the stack only stores non-terminals: $\Gamma = V$

From Context-Free Grammars to PDMs

- Assume we have G a normalized CFG: for every $A \to w \in P$, w is either a single terminal or a string of non-terminals
- We construct a PDM M such that $L(M) = L(G)$:



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

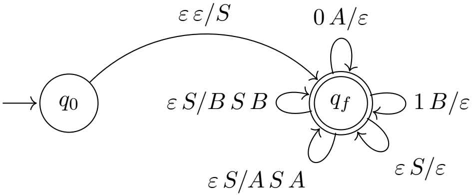
$$[q_f, w, S] \vdash^* [q_f, v, \alpha] \equiv \exists u \in \Sigma^* : w = uv \wedge S \Rightarrow^*_{lm} u\alpha$$

From CFGs to PDMs: Example

Given the normalized grammar

$$S \to A\,S\,A \mid B\,S\,B \mid \varepsilon \qquad A \to 0 \qquad B \to 1$$

We have the following extended PDM:

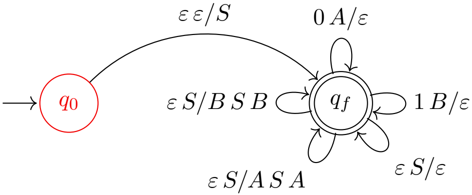


From CFGs to PDMs: Example

Given the normalized grammar

$$S \to A\,S\,A \mid B\,S\,B \mid \varepsilon \qquad A \to 0 \qquad B \to 1$$

We have the following extended PDM:



We can check:

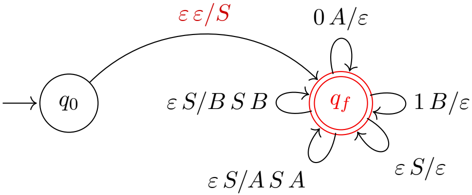
$$[q_0, 1001, \varepsilon] \vdash$$

From CFGs to PDMs: Example

Given the normalized grammar

$$S \to A\,S\,A \mid B\,S\,B \mid \varepsilon \qquad A \to 0 \qquad B \to 1$$

We have the following extended PDM:



We can check:

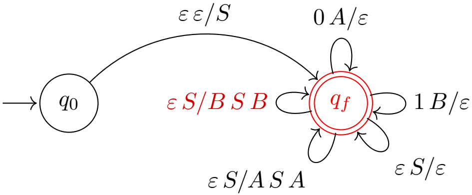
$$[q_0, 1001, \varepsilon] \vdash [q_1, 1001, S] \vdash$$

From CFGs to PDMs: Example

Given the normalized grammar

$$S \to A\,S\,A \mid B\,S\,B \mid \varepsilon \qquad A \to 0 \qquad B \to 1$$

We have the following extended PDM:



We can check:

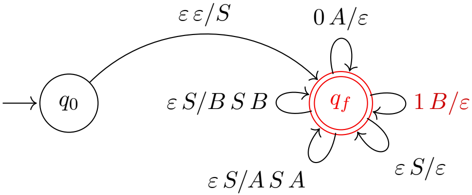
$$[q_0, 1001, \varepsilon] \vdash [q_1, 1001, S] \vdash [q_1, 1001, B\,S\,B] \vdash$$

From CFGs to PDMs: Example

Given the normalized grammar

$$S \to A\,S\,A \mid B\,S\,B \mid \varepsilon \qquad A \to 0 \qquad B \to 1$$

We have the following extended PDM:



We can check:

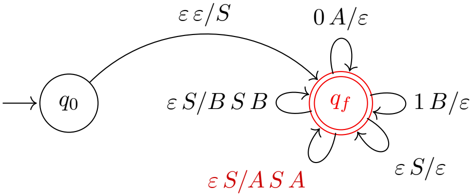
$$[q_0, 1001, \varepsilon] \vdash [q_1, 1001, S] \vdash [q_1, 1001, B\,S\,B] \vdash$$
$$[q_1, 001, S\,B] \vdash$$

From CFGs to PDMs: Example

Given the normalized grammar

$$S \to A\,S\,A \mid B\,S\,B \mid \varepsilon \qquad A \to 0 \qquad B \to 1$$

We have the following extended PDM:



We can check:

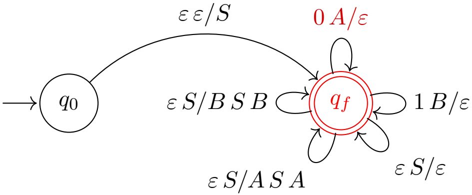
$$[q_0, 1001, \varepsilon] \vdash [q_1, 1001, S] \vdash [q_1, 1001, B\,S\,B] \vdash$$
$$[q_1, 001, S\,B] \vdash [q_1, 001, A\,S\,A\,B] \vdash$$

From CFGs to PDMs: Example

Given the normalized grammar

$$S \to A\,S\,A \mid B\,S\,B \mid \varepsilon \qquad A \to 0 \qquad B \to 1$$

We have the following extended PDM:



We can check:

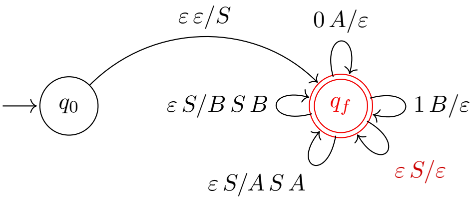
$$[q_0, 1001, \varepsilon] \vdash [q_1, 1001, S] \vdash [q_1, 1001, B\,S\,B] \vdash$$
$$[q_1, 001, S\,B] \vdash [q_1, 001, A\,S\,A\,B] \vdash [q_1, 01, S\,A\,B] \vdash$$

From CFGs to PDMs: Example

Given the normalized grammar

$$S \to A\,S\,A \mid B\,S\,B \mid \varepsilon \qquad A \to 0 \qquad B \to 1$$

We have the following extended PDM:



We can check:

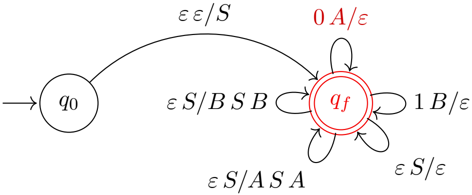
$$[q_0, 1001, \varepsilon] \vdash [q_1, 1001, S] \vdash [q_1, 1001, B\,S\,B] \vdash$$
$$[q_1, 001, S\,B] \vdash [q_1, 001, A\,S\,A\,B] \vdash [q_1, 01, S\,A\,B] \vdash$$
$$[q_1, 01, A\,B] \vdash$$

From CFGs to PDMs: Example

Given the normalized grammar

$$S \to A\,S\,A \mid B\,S\,B \mid \varepsilon \qquad A \to 0 \qquad B \to 1$$

We have the following extended PDM:



We can check:

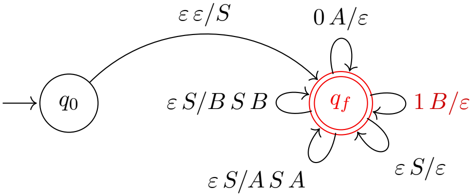
$$[q_0, 1001, \varepsilon] \vdash [q_1, 1001, S] \vdash [q_1, 1001, B\,S\,B] \vdash$$
$$[q_1, 001, S\,B] \vdash [q_1, 001, A\,S\,A\,B] \vdash [q_1, 01, S\,A\,B] \vdash$$
$$[q_1, 01, A\,B] \vdash [q_1, 1, B] \vdash$$

From CFGs to PDMs: Example

Given the normalized grammar

$$S \to A\,S\,A \mid B\,S\,B \mid \varepsilon \qquad A \to 0 \qquad B \to 1$$

We have the following extended PDM:



We can check:

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

From PDMs to CFGs

Idea: given a machine M, non-terminals of the form S or $\langle q, A, r \rangle$ for $q, r \in Q, A \in \Gamma \cup \{\varepsilon\}$

$\langle q, A, r \rangle \Rightarrow w \quad \sim \quad M$ can read w from the state q with the stack A, and end with the empty stack in r

Add the following production rules:

1. $S \to \langle q_0, \varepsilon, q_f \rangle$
2. $\langle q, A, r \rangle \to a \langle p, B, r \rangle$
3. $\langle q, A, r \rangle \to \langle q, \varepsilon, p \rangle \langle p, A, r \rangle$
4. $\langle q, \varepsilon, q \rangle \to \varepsilon$

From PDMs to CFGs

Idea: given a machine M, non-terminals of the form S or $\langle q, A, r\rangle$ for $q, r \in Q, A \in \Gamma \cup \{\varepsilon\}$

$\langle q, A, r\rangle \Rightarrow w \qquad \sim \qquad$ M can read w from the state q with the stack A, and end with the empty stack in r

For every $q_f \in F$

Add the following production rules:

1. $S \to \langle q_0, \varepsilon, q_f\rangle$
2. $\langle q, A, r\rangle \to a\langle p, B, r\rangle$
3. $\langle q, A, r\rangle \to \langle q, \varepsilon, p\rangle\langle p, A, r\rangle$
4. $\langle q, \varepsilon, q\rangle \to \varepsilon$

From PDMs to CFGs

Idea: given a machine M, non-terminals of the form S or $\langle q, A, r\rangle$ for $q, r \in Q, A \in \Gamma \cup \{\varepsilon\}$

$\langle q, A, r\rangle \Rightarrow w \quad \sim \quad M$ can read w from the state q with the stack A, and end with the empty stack in r

For every $q_f \in F$

Add the following production rules:

1. $S \to \langle q_0, \varepsilon, q_f\rangle$
2. $\langle q, A, r\rangle \to a\langle p, B, r\rangle$
3. $\langle q, A, r\rangle \to \langle q, \varepsilon, p\rangle\langle p, A, r\rangle$
4. $\langle q, \varepsilon, q\rangle \to \varepsilon$

Use A now for the transition $p \xrightarrow{a\ A/B} q$

From PDMs to CFGs

Idea: given a machine M, non-terminals of the form S or $\langle q, A, r\rangle$ for $q, r \in Q, A \in \Gamma \cup \{\varepsilon\}$

$\langle q, A, r\rangle \Rightarrow w \quad \sim \quad$ M can read w from the state q with the stack A, and end with the empty stack in r

For every $q_f \in F$

Add the following production rules:

1. $S \to \langle q_0, \varepsilon, q_f\rangle$
2. $\langle q, A, r\rangle \to a\langle p, B, r\rangle$
3. $\langle q, A, r\rangle \to \langle q, \varepsilon, p\rangle\langle p, A, r\rangle$
4. $\langle q, \varepsilon, q\rangle \to \varepsilon$

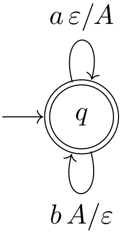
Use A now for the transition $p \xrightarrow{a\ A/B} q$

Use A later from an intermediate p

Done processing

Example

1. $S \to \langle q, \varepsilon, q\rangle$
2. $\langle q, \varepsilon, q\rangle \to \varepsilon \mid a\langle q, A, q\rangle$
3. $\langle q, A, q\rangle \to b\langle q, \varepsilon, q\rangle \mid \langle q, \varepsilon, q\rangle\langle q, A, q\rangle$



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

- 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).

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 for CFLs

We didn't cover (self study!):

- Pumping Lemma for CFLs (Sect 4.2)

Reading: Reader: 4.1-4.3; Kozen: 19-25; Sudkamp: 7.1-7.5.

Next Lecture(s): Turing Machines