Pushdown Automata

COMP2600 — Formal Methods for Software Engineering

Katya Lebedeva

Australian National University
Semester 2, 2014
Pushdown Automata — PDA

Finite State Control

Stack memory

read head

input tape

a0 a1 a2 ... an

COMP 2600 — Pushdown Automata
Intuition

A finite state control reads the string, one symbol at a time (the input symbol).

Informally, a PDA determines its transition by observing

- the input symbol
- its current state
- the symbol on the top of stack

Alternatively, it may make a spontaneous transition using $\varepsilon$ as its input instead of an input symbol!
A Transition of a PDA

In one transition the PDA does the following:

1. Consumes the input symbol from the input string. If $\epsilon$ is the input symbol, then no input symbol is consumed.

2. Goes to a new state. It may be the same state.

3. Replaces the symbol at the top of the stack by any string (see next slide!!!)
Replacing the top symbol $A$ of the stack by a string $\gamma$

Let $\Gamma$ be the stack alphabet (i.e. the set of stack symbols).
Let $A \in \Gamma$ be the top symbol on the stack.
Let $B \in \Gamma$ and $B \neq A$.
Let $w = X_1X_2 \ldots X_n$, $w \in \Gamma^*$. 

<table>
<thead>
<tr>
<th>$\gamma$</th>
<th>Operation on stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>Popping: removal of the most recently added element (stack’s top symbol)</td>
</tr>
<tr>
<td>$A$</td>
<td>Popping stack’s top symbol. Pushing $A$. Hence, no change is made.</td>
</tr>
<tr>
<td>$B$</td>
<td>Popping stack’s top symbol. Pushing $B$.</td>
</tr>
<tr>
<td>$w$</td>
<td>Popping stack’s top symbol. Pushing $X_1X_2 \ldots X_n$ onto the stack ($X_1$ is on the top).</td>
</tr>
</tbody>
</table>
Example

\[\{a^n b^n \mid n \geq 1\}\]

Recall that this language cannot be recognised by a FSA.

It can be recognised by a PDA!

Intuition:

- **begin** in state \(q_0\) with the start symbol \(Z\) on the stack
- **phase 1**: (state \(q_1\)) push \(a\)’s, one by one, from the input onto the stack
- **phase 2**: (state \(q_2\)) when seeing \(b\) on the tape pop \(a\) from the stack
- **finalise**: if the stack is empty and the input is exhausted in the final state (\(q_3\)), accept the string.
The stack starts off containing only the initial symbol $Z$.

The transition function of the PDA to recognise $a^n b^n$ is as follows:

\[
\delta(q_0, a, Z) = \{(q_1, aZ)\} \quad \cdots \quad \text{push first } a
\]
\[
\delta(q_1, a, a) = \{(q_1, aa)\} \quad \cdots \quad \text{push further } a's
\]
\[
\delta(q_1, b, a) = \{(q_2, \varepsilon)\} \quad \cdots \quad \text{start popping } a's
\]
\[
\delta(q_2, b, a) = \{(q_2, \varepsilon)\} \quad \cdots \quad \text{pop further } a's
\]
\[
\delta(q_2, \varepsilon, Z) = \{(q_3, \varepsilon)\} \quad \cdots \quad \text{accept}
\]

where $q_0$ is the start state and $q_3$ is the final state (underlined).
A Transition Diagram for PDA’s

- The nodes correspond to the states of the PDA.
- An arrow labelled start indicates the start state. Doubly circled states are accepting.
- If $\delta(q, a, X)$ contains a pair $(p, \alpha)$, then there is an arc from $q$ to $p$ labeled $a, X/\alpha$.

The only thing that the diagram does not tell us is which stack symbol is the start symbol. Conventionally, it is $Z$. 

The diagram to our PDA for $\{a^n b^n \mid n \geq 1\}$

\[
\begin{align*}
\delta(q_0, a, Z) &= \{(q_1, aZ)\} \quad \cdots \text{push first } a \\
\delta(q_1, a, a) &= \{(q_1, aa)\} \quad \cdots \text{push further } a's \\
\delta(q_1, b, a) &= \{(q_2, \varepsilon)\} \quad \cdots \text{start popping } a's \\
\delta(q_2, b, a) &= \{(q_2, \varepsilon)\} \quad \cdots \text{pop further } a's \\
\delta(q_2, \varepsilon, Z) &= \{(q_3, \varepsilon)\} \quad \cdots \text{accept}
\end{align*}
\]
Transition function

Transition function of a determinstic PDA:

\[ \delta_D : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma^* \]

\[ \delta_D : (\text{state, input symbol or } \varepsilon, \text{top-of-stack}) \rightarrow (\text{new state, string of stack's symbols}) \]

Note that \( \delta_D \) can be partial.

Transition function of a non-deterministic PDA:

\[ \delta_N : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*} \]

\[ \delta_N : (\text{state, input symbol or } \varepsilon, \text{top-of-stack}) \rightarrow \text{set of (new state, string of stack's symbols)} \]
Definition of a Nondeterministic PDA

A nondeterministic PDA has the form \((Q, q_0, F, \Sigma, \Gamma, Z, \delta)\), where

- \(Q\) is the set of states
  \(q_0 \in Q\) is the initial state and \(F \subseteq Q\) is the set of the final states
- \(\Sigma\) is the set of input symbols (the alphabet)
- \(\Gamma\) is the set of stack symbols (the stack alphabet)
  \(Z \in \Gamma\) is the initial stack symbol
- \(\delta\) is a transition function
  \[\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}\]
Definition of a Deterministic PDA

A deterministic PDA has the form \((Q, q_0, F, \Sigma, \Gamma, Z, \delta)\), where

- \(Q\) is the set of states
  - \(q_0 \in Q\) is the initial state and \(F \subseteq Q\) is the set of the final states
- \(\Sigma\) is the set of input symbols (the alphabet)
- \(\Gamma\) is the set of stack symbols (the stack alphabet)
  - \(Z \in \Gamma\) is the initial stack symbol
- \(\delta\) is a (partial) transition function
  \[
  \delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*
  \]
  such that for all \(q \in Q\) and \(s \in \Gamma\), \(\delta(q, \varepsilon, s)\) is defined iff \(\delta(q, a, s)\) is undefined for all \(a \in \Sigma\).
PDA execution: reading a symbol $x \in \Sigma$ or $\varepsilon \notin \Sigma$

- $(q_2, \sigma) \in \delta(q_1, x, Y)$

<table>
<thead>
<tr>
<th>state</th>
<th>tape’s head</th>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>points at $x$</td>
<td>$Y$ on top-of-stack</td>
</tr>
<tr>
<td>$q_2$</td>
<td>advances</td>
<td>$\sigma$ replaces $Y$ on stack</td>
</tr>
</tbody>
</table>

- $(q_2, \sigma) \in \delta(q_1, \varepsilon, Y)$

<table>
<thead>
<tr>
<th>state</th>
<th>tape’s head</th>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>points any symbol from $\Sigma$</td>
<td>$Y$ on top-of-stack</td>
</tr>
<tr>
<td>$q_2$</td>
<td>does not advance</td>
<td>$\sigma$ replaces $Y$ on stack</td>
</tr>
</tbody>
</table>

$\varepsilon$-transitions go from one state to another spontaneously, leaving the input symbol intact (the tape head is not advanced).
Instantaneous Descriptions of a PDA

PDA goes from configuration to configuration in response to input symbols.

PDA’s configuration involves both state and content of the stack.

It is also useful to see the portion of the input string that remains.

We shall represent the configuration of a PDA as a triple \((q, w, \gamma)\), called an instantaneous description:

- \(q\) is the state
- \(w\) is the remaining input
- \(\gamma\) is the stack contents

**Convention:** show the top of the stack at the left end of \(\gamma\) and the bottom at the right end
Representing one computation step of a PDA

Let \( P = (Q, q_0, F, \Sigma, \Gamma, Z, \delta) \) be a PDA.

Suppose \( \delta(q, a, X) \) contains \( (p, \alpha) \).
Then for all \( w \in \Sigma^* \) and \( \beta \in \Gamma^* \)

\[
(q, aw, X\beta) \vdash (p, w, \alpha\beta)
\]

Consuming \( a \) (which may be \( \varepsilon \)) from the input and replacing \( X \) on top of the stack by \( \alpha \), we can go from state \( q \) to state \( p \).

Note that what remains on the input, i.e. \( w \), and what is below the top of the stack, i.e. \( \beta \), do not influence the action of the PDA in this computation step.

They are merely carried along and can influence later steps.
Example

Consider the input string $aaabbb$.

$$(q_0, aaabbb, Z) \vdash (q_1, aabbb, aZ)$$  \hspace{1cm} \text{(push first a)}

$$\vdash (q_1, abbb, aaZ)$$  \hspace{1cm} \text{(push further a's)}

$$\vdash (q_1, bbb, aaaZ)$$  \hspace{1cm} \text{(push further a's)}

$$\vdash (q_2, bb, aaZ)$$  \hspace{1cm} \text{(start popping a's)}

$$\vdash (q_2, b, aZ)$$  \hspace{1cm} \text{(pop further a's)}

$$\vdash (q_2, \epsilon, Z)$$  \hspace{1cm} \text{(pop further a's)}

$$\vdash (q_3, \epsilon, \epsilon)$$  \hspace{1cm} \text{(accept)}

The machine halts in the final state with input exhausted and an empty stack. The string is accepted.
**Computation of a PDA**

\((Q, q_0, F, \Sigma, \Gamma, Z, \delta)\)

**Start configuration:**

- PDA is in state \(q_0\)
- the tape head is on the leftmost symbol of the input string
- the stack contains only \(Z\)

**Computation and termination:** Starting at the start configuration, the PDA performs a sequence of computation steps (moves). It terminates when the stack becomes empty.

**Acceptance by final state:** The PDA consumes the input and enters a final state.

**Acceptance by empty stack:** The PDA consumes the input and empties the stack.
Example: Palindromes with ‘Centre Mark’

Consider the language

$$\{wcw^R \mid w \in \{a, b\}^* \land w^R \text{ is } w \text{ reversed}\}$$

This language is context-free, and we can design a deterministic PDA to accept it:

- Push $a$’s and $b$’s onto the stack as we see them
- When we see $c$, change state
- Now try to match the symbols we are reading with the symbols on top of the stack, popping as we go
- If the top of the stack has symbol $Z$, pop it and enter the final state via an $\varepsilon$-transition. Hopefully our input has been used up too!

Full formal details are left as an exercise.
Languages of Nondeterministic and Deterministic PDAs.

Nondeterministic PDAs recognise context free languages.

Deterministic PDAs recognise all the regular languages, but only a proper subset of context free languages!

\[ L(DPDA) \subset L(NPDA) \]
Example: Even-Length Palindromes

Consider the context free language of even length palindromes

\[ \{ww^R \mid w \in \{a, b\}^* \land w^R \text{ is } w \text{ reversed}\} \]

It cannot be recognised by a deterministic PDA, because without a centre mark it cannot know whether it is in the first half of a sentence (and should continue pushing into memory) or the second half (and should be matching input and stack, and popping).

But a non-deterministic PDA can recognise this language.
\[ L = \{ ww^R \mid w \in \{a, b\}^* \land w^R \text{ is } w \text{ reversed} \} \]

The following transitions are necessary to recognize \( L \)

\[
(r, Z) \in \delta(q, \varepsilon, Z)
\]

\[
(r, a) \in \delta(q, \varepsilon, a)
\]

\[
(r, b) \in \delta(q, \varepsilon, b)
\]

\( q \) is the ‘push’ state, and \( r \) the ‘match and pop’ state.

In other words, we continually ‘guess’ which job we should be doing!

Task: Define other transitions!
Grammars and PDAs

Theorem

The class of languages recognised by non-deterministic PDA’s is exactly the class of context-free languages.

We will only justify this result in one direction: for any CFG, there is a corresponding PDA.

This direction is the basis of automatically deriving parsers from grammars.
From CFG to PDA

Given a CFG

\[ G = (\Sigma, N, S, P) \]

We define a PDA

\[ P = (\{q_0, q_1, q_2\}, q_0, \{q_2\}, \Sigma, N \cup \Sigma, Z, \delta) \]

Note that \( P \) has three states: \( q_0 \) (initial), \( q_1 \) (processing), and \( q_2 \) (final). Its alphabet the alphabet (i.e. the set of terminals) of \( G \), and the set of its stack symbols consists of \( G \)'s terminals and non-terminals.

\( P \) is a non-deterministic PDA since there may be several productions for each non-terminal.
1. Initialise the process by pushing the start symbol $S$ onto the stack, and entering state $q_1$:

$$
\delta(q_0, \varepsilon, Z) = \{(q_1, SZ)\}
$$

2. For each production $A \rightarrow \alpha$ define

$$(q_1, \alpha) \in \delta(q_1, \varepsilon, A)$$

Thus, if a non-terminal is on top of stack, it is replaced it with the right hand side of the production.

3. For each terminal symbol $t$, pop the stack if it matches the input:

$$
\delta(q_1, t, t) = \{(q_1, \varepsilon)\}
$$

4. For termination, add the transition, with final state $q_2$:

$$
\delta(q_1, \varepsilon, Z) = \{(q_2, \varepsilon)\}$$
Example

\[
S \rightarrow S + T \mid T \\
T \rightarrow T \ast U \mid U \\
U \rightarrow (S) \mid i
\]

1. Initialise:
\[
\delta(q_0, \varepsilon, Z) = \{(q_1, SZ)\}
\]

2. Expand non-terminals:
\[
\delta(q_1, \varepsilon, S) = \{(q_1, S + T), (q_1, T)\} \\
\delta(q_1, \varepsilon, T) = \{(q_1, T \ast U), (q_1, U)\} \\
\delta(q_1, \varepsilon, U) = \{(q_1, (S)), (q_1, i)\}
\]
3. Match and pop terminals:

$$\delta(q_1, +, +) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, *, *) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, i, i) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, (, ) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, ), ) = \{(q_1, \varepsilon)\}$$

4. Terminate:

$$\delta(q_1, \varepsilon, Z) = \{(q_2, \varepsilon)\}$$
IDs on input $i^*i$

$$(q_0, i^*i, Z) \vdash (q_1, i^*i, SZ)$$

$$(q_1, i^*i, TZ)$$

$$(q_1, i^*i, T^*UZ)$$

$$(q_1, i^*i, U^*UZ)$$

$$(q_1, i^*i, i^*UZ)$$

$$(q_1, ^*i, ^*UZ)$$

$$(q_1, i, UZ)$$

$$(q_1, i, iZ)$$

$$(q_1, \varepsilon, Z)$$

$$(q_1, \varepsilon, Z)$$

accept by empty stack and final state