Basics of Pushdown Automata

Languages, specified by context free grammars are accepted by Pushdown Automata:

The pushdown is used e.g. to verify correct nesting of braces.
Example:

States: 0, 1, 2
Start state: 0
Final states: 0, 2

Conventions:
- We do not differentiate between pushdown symbols and states.
- The rightmost/upper pushdown symbol represents the state.
- Every transition consumes/modifies the upper part of the pushdown.

Pushdown Automata

Definition:
A pushdown automaton (PDA) is a tuple $M = (Q, T, \delta, q_0, F)$ with:
- $Q$: a finite set of states;
- $T$: an input alphabet;
- $q_0 \in Q$: the start state;
- $F \subseteq Q$: the set of final states and
- $\delta \subseteq Q^* \times (T \cup \{\varepsilon\}) \times Q^*$: a finite set of transitions.

We define computations of pushdown automata with the help of transitions; a particular computation state (the current configuration) is a pair:

$(\gamma, w) \in Q^* \times T^*$

consisting of the pushdown content and the remaining input.

... for example:

States: 0, 1, 2
Start state: 0
Final states: 0, 2

0 $\rightarrow$ a
1 $\rightarrow$ a
11 $\rightarrow$ 2
12 $\rightarrow$ 2
... for example:

<table>
<thead>
<tr>
<th>States:</th>
<th>0, 1, 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start state:</td>
<td>0</td>
</tr>
<tr>
<td>Final states:</td>
<td>0, 2</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
</tbody>
</table>

\( (0, \text{aabb}) \vdash (11, \text{aabb}) \)

... for example:

<table>
<thead>
<tr>
<th>States:</th>
<th>0, 1, 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start state:</td>
<td>0</td>
</tr>
<tr>
<td>Final states:</td>
<td>0, 2</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
</tbody>
</table>

\( (0, \text{aabb}) \vdash (11, \text{aabb}) \)

\( \vdash (111, \text{aabb}) \)

\( \vdash (1111, \text{aabb}) \)
... for example:

<table>
<thead>
<tr>
<th>States:</th>
<th>0, 1, 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>11</td>
<td>b</td>
</tr>
<tr>
<td>12</td>
<td>b</td>
</tr>
<tr>
<td>Start state:</td>
<td>0</td>
</tr>
<tr>
<td>Final states:</td>
<td>0, 2</td>
</tr>
</tbody>
</table>

\[
(0, \text{a}a\text{b}b\text{b}) \vdash (1, \text{a}a\text{b}b\text{b}) \\
\vdash (1, \text{a}b\text{b}b) \\
\vdash (11, \text{a}b\text{b}b) \\
\vdash (1111, \text{b}b\text{b}) \\
\vdash (1, \text{b}b) \\
\vdash (12, \text{b})
\]

A computation step is characterized by the relation
\[
\vdash \subseteq (Q^* \times T^*)^2 \text{ with }
\]
\[
(\alpha \gamma, xw) \vdash (\alpha \gamma', w) \text{ for } (\gamma, x, \gamma') \in \delta
\]

Remarks:
- The relation \(\vdash\) depends on the pushdown automaton \(M\)
- The reflexive and transitive closure of \(\vdash\) is called \(\vdash^*\)
- Then, the language, accepted by \(M\), is

\[
\mathcal{L}(M) = \{w \in T^* | \exists f \in F : (q_0, w) \vdash^* (f, \epsilon)\}
\]
Deterministic Pushdown Automaton

Definition:
The pushdown automaton \( M \) is deterministic, if every
configuration has maximally one successor configuration.

This is exactly the case if for distinct transitions
\((γ_1, x, γ_2), (γ'_1, x', γ'_2) \in δ\) we can assume:
Is \( γ_1 \) a suffix of \( γ'_1 \), then \( x \neq x' \land x \neq ε \neq x' \) is valid.

... for example:

<table>
<thead>
<tr>
<th>0</th>
<th>a</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>b</td>
<td>2</td>
</tr>
</tbody>
</table>

... this obviously holds

Pushdown Automata

Theorem:
For each context free grammar \( G = (N, T, P, S) \)
a pushdown automaton \( M \) with \( \mathcal{L}(G) = \mathcal{L}(M) \) can be built.

The theorem is so important for us, that we take a look at two
constructions for automata, motivated by both of the special
derivations:

- \( M'_G \) to build Leftmost derivations
- \( M''_G \) to build reverse Rightmost derivations

Item Pushdown Automaton

Construction:  Item Pushdown Automaton \( M'_G \)

- Reconstruct a Leftmost derivation.
- Expand nonterminals using a rule.
- Verify successively, that the chosen rule matches the input.

\[ [A \rightarrow α \bullet β], \quad A \rightarrow α β \in P \]

The states are now Items (= rules with a dot):

The dot marks the spot, how far the rule is already processed
Our example:

\[ S \rightarrow AB \quad A \rightarrow a \quad B \rightarrow b \]
Item Pushdown Automaton – Example

Our example:

\[ S \rightarrow AB \quad A \rightarrow a \quad B \rightarrow b \]

Item Pushdown Automaton – Example

We add another rule \( S' \rightarrow S \) for initialising the construction:

Start state: \( [S' \rightarrow \bullet S] \)
End state: \( [S' \rightarrow S \bullet] \)

Transition relations:

\[
\begin{align*}
[S' \rightarrow \bullet S] & \quad \epsilon & [S' \rightarrow \bullet S] & \quad \epsilon & [S \rightarrow \bullet AB] \\
[S \rightarrow \bullet AB] & \quad \epsilon & [S \rightarrow \bullet AB] & \quad \epsilon & [A \rightarrow \bullet a] \\
A \rightarrow \bullet a & \quad a & A \rightarrow \bullet a & \\
[S \rightarrow \bullet AB] & \quad \epsilon & [S \rightarrow \bullet AB] & \quad \epsilon & [A \rightarrow a \bullet] \\
[S \rightarrow \bullet AB] & \quad \epsilon & [S \rightarrow \bullet AB] & \quad \epsilon & [B \rightarrow \bullet b] \\
B \rightarrow \bullet b & \quad b & B \rightarrow \bullet b & \\
[S \rightarrow \bullet AB] & \quad \epsilon & [S \rightarrow \bullet AB] & \quad \epsilon & [S \rightarrow \bullet AB] \\
[S' \rightarrow \bullet S] & \quad \epsilon & [S \rightarrow \bullet AB] & \quad \epsilon & [S' \rightarrow S \bullet]
\end{align*}
\]

Item Pushdown Automaton – Example

Our example:

\[ S \rightarrow AB \quad A \rightarrow a \quad B \rightarrow b \]

Item Pushdown Automaton – Example

We add another rule \( S' \rightarrow S \) for initialising the construction:

Start state: \( [S' \rightarrow \bullet S] \)
End state: \( [S' \rightarrow S \bullet] \)

Transition relations:

\[
\begin{align*}
[S' \rightarrow \bullet S] & \quad \epsilon & [S' \rightarrow \bullet S] & \quad \epsilon & [S \rightarrow \bullet AB] \\
[S \rightarrow \bullet AB] & \quad \epsilon & [S \rightarrow \bullet AB] & \quad \epsilon & [A \rightarrow \bullet a] \\
A \rightarrow \bullet a & \quad a & A \rightarrow \bullet a & \\
[S \rightarrow \bullet AB] & \quad \epsilon & [S \rightarrow \bullet AB] & \quad \epsilon & [A \rightarrow a \bullet] \\
[S \rightarrow \bullet AB] & \quad \epsilon & [S \rightarrow \bullet AB] & \quad \epsilon & [B \rightarrow \bullet b] \\
B \rightarrow \bullet b & \quad b & B \rightarrow \bullet b & \\
[S \rightarrow \bullet AB] & \quad \epsilon & [S \rightarrow \bullet AB] & \quad \epsilon & [S \rightarrow \bullet AB] \\
[S' \rightarrow \bullet S] & \quad \epsilon & [S \rightarrow \bullet AB] & \quad \epsilon & [S' \rightarrow S \bullet]
\end{align*}
\]
Item Pushdown Automaton

The item pushdown automaton $M^L_G$ has three kinds of transitions:

Expansions: $([A \rightarrow \alpha \bullet B \beta], \epsilon, [A \rightarrow \alpha \bullet B \beta] [B \rightarrow \bullet \gamma])$ for $A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P$

Shifts: $([A \rightarrow \alpha \bullet a \beta], a, [A \rightarrow \alpha a \bullet \beta])$ for $A \rightarrow \alpha a \beta \in P$

Reduces: $([A \rightarrow \alpha \bullet B \beta] [B \rightarrow \gamma \bullet], \epsilon, [A \rightarrow \alpha B \bullet \beta])$ for $A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P$

Items of the form: $[A \rightarrow \alpha \bullet \beta]$ are also called complete

The item pushdown automaton shifts the dot once around the derivation tree ...

Discussion:

- The expansions of a computation form a leftmost derivation
- Unfortunately, the expansions are chosen nondeterministically
- For proving correctness of the construction, we show that for every item $[A \rightarrow \alpha \bullet B \beta]$ the following holds:
  $([A \rightarrow \alpha \bullet B \beta], w) \vdash^* ([A \rightarrow \alpha B \bullet \beta], \epsilon)$ iff $B \vdash^* w$
- LL-Parsing is based on the item pushdown automaton and tries to make the expansions deterministic ...

Item Pushdown Automaton

The item pushdown automaton $M^L_G$ has three kinds of transitions:

Expansions: $([A \rightarrow \alpha \bullet B \beta], \epsilon, [A \rightarrow \alpha \bullet B \beta] [B \rightarrow \bullet \gamma])$ for $A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P$

Shifts: $([A \rightarrow \alpha \bullet a \beta], a, [A \rightarrow \alpha a \bullet \beta])$ for $A \rightarrow \alpha a \beta \in P$

Reduces: $([A \rightarrow \alpha \bullet B \beta] [B \rightarrow \gamma \bullet], \epsilon, [A \rightarrow \alpha B \bullet \beta])$ for $A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P$

Items of the form: $[A \rightarrow \alpha \bullet \beta]$ are also called complete

The item pushdown automaton shifts the dot once around the derivation tree ...

Item Pushdown Automaton

The item pushdown automaton $M^L_G$ has three kinds of transitions:

Expansions: $([A \rightarrow \alpha \bullet B \beta], \epsilon, [A \rightarrow \alpha \bullet B \beta] [B \rightarrow \bullet \gamma])$ for $A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P$

Shifts: $([A \rightarrow \alpha \bullet a \beta], a, [A \rightarrow \alpha a \bullet \beta])$ for $A \rightarrow \alpha a \beta \in P$

Reduces: $([A \rightarrow \alpha \bullet B \beta] [B \rightarrow \gamma \bullet], \epsilon, [A \rightarrow \alpha B \bullet \beta])$ for $A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P$

Items of the form: $[A \rightarrow \alpha \bullet \beta]$ are also called complete

The item pushdown automaton shifts the dot once around the derivation tree ...
**Item Pushdown Automaton**

Beispiel: $S \rightarrow \epsilon \mid aSb$

The transitions of the according Item Pushdown Automaton:

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$S' \rightarrow \epsilon$</td>
<td>$S \rightarrow \epsilon$</td>
</tr>
<tr>
<td>1</td>
<td>$S' \rightarrow \epsilon$</td>
<td>$S \rightarrow \epsilon$</td>
</tr>
<tr>
<td>2</td>
<td>$S \rightarrow \epsilon aSb$</td>
<td>$a$ $S \rightarrow a \epsilon Sb$</td>
</tr>
<tr>
<td>3</td>
<td>$S \rightarrow a \epsilon Sb$</td>
<td>$\epsilon$ $S \rightarrow a \epsilon Sb$</td>
</tr>
<tr>
<td>4</td>
<td>$S \rightarrow a \epsilon Sb$</td>
<td>$\epsilon$ $S \rightarrow a \epsilon Sb$</td>
</tr>
<tr>
<td>5</td>
<td>$S \rightarrow a \epsilon Sb$</td>
<td>$\epsilon$ $S \rightarrow a \epsilon Sb$</td>
</tr>
<tr>
<td>6</td>
<td>$S \rightarrow a \epsilon Sb$</td>
<td>$\epsilon$ $S \rightarrow a \epsilon Sb$</td>
</tr>
<tr>
<td>7</td>
<td>$S \rightarrow a \epsilon Sb$</td>
<td>$\epsilon$ $S \rightarrow a \epsilon Sb$</td>
</tr>
<tr>
<td>8</td>
<td>$S \rightarrow a \epsilon Sb$</td>
<td>$\epsilon$ $S \rightarrow a \epsilon Sb$</td>
</tr>
</tbody>
</table>

Conflicts arise between the transitions (0, 1) and (3, 4), resp.

**Topdown Parsing**

**Problem:**
Conflicts between the transitions prohibit an implementation of the item pushdown automaton as deterministic pushdown automaton.

**Idee 1: GLL Parsing**
For each conflict, we create a virtual copy of the complete stack and continue computing in parallel.

**Topdown Parsing**

**Problem:**
Conflicts between the transitions prohibit an implementation of the item pushdown automaton as deterministic pushdown automaton.

**Idee 1: GLL Parsing**
For each conflict, we create a virtual copy of the complete stack and continue computing in parallel.

**Idee 2: Recursive Descent & Backtracking**
Depth-first search for an appropriate solution.
Topdown Parsing

**Problem:**
Conflicts between the transitions prohibit an implementation of the item pushdown automaton as deterministic pushdown automaton.

**Idee 1:** GLL Parsing
For each conflict, we create a virtual copy of the complete stack and continue computing in parallel.

**Idee 2:** Recursive Descent & Backtracking
Depth-first search for an appropriate solution.

**Idee 3:** Recursive Descent & Lookahead
Conflicts are resolved by considering a lookup of the next input symbol.

Structure of the $LL(1)$-Parser:

- The parser accesses a frame of length 1 of the input;
- it corresponds to an item pushdown automaton, essentially;
- table $M[q, w]$ contains the rule of choice.

Topdown Parsing

**Idee:**
- Emanate from the item pushdown automaton
- Consider the next symbol to determine the appropriate rule for the next expansion
  - A grammar is called $LL(1)$ if a unique choice is always possible

**Definition:**
A reduced grammar is called $LL(1)$ if for each two distinct rules $A \rightarrow \alpha$, $A \rightarrow \alpha'$ in $P$ and each derivation $S \rightarrow^*_T u A \beta$ with $u \in T^*$ the following is valid:

$$\text{First}(\alpha \beta) \cap \text{First}(\alpha' \beta) = \emptyset$$
**Topdown Parsing**

Example 1:

\[
S \rightarrow \text{if } (E) S \text{ else } S \mid \\
\text{while } (E) S \mid \\
E;
\]

\[
E \rightarrow \text{id}
\]

is LL(1), since \(\text{First}_1(E) = \{\text{id}\}\)

**Topdown Parsing**

Example 1:

\[
S \rightarrow \text{if } (E) S \text{ else } S \mid \\
\text{while } (E) S \mid \\
E;
\]

\[
E \rightarrow \text{id}
\]

is LL(1), since \(\text{First}_1(E) = \{\text{id}\}\)

Example 2:

\[
S \rightarrow \text{if } (E) S \text{ else } S \mid \\
\text{if } (E) S \mid \\
\text{while } (E) S \mid \\
E;
\]

\[
E \rightarrow \text{id}
\]

... is not LL(k) for any \(k > 0\).

**Lookahead Sets**

**Definition:**

For a set \(L \subseteq T^*\) we define:

\[
\text{First}_1(L) = \{e | e \in L\} \cup \{u \in T \mid \exists v \in T^*: uv \in L\}
\]

**Lookahead Sets**

**Definition:**

For a set \(L \subseteq T^*\) we define:

\[
\text{First}_1(L) = \{e | e \in L\} \cup \{u \in T \mid \exists v \in T^*: uv \in L\}
\]

**Example:**

- \(e\)
- \(ab\)
- \(aabb\)
- \(aa\ aabb\)

the prefixes of length 1
Lookahead Sets

**Arithmetics:**
First$_1(\emptyset)$ is **compatible** with union and concatenation:

- First$_1(\emptyset) = \emptyset$
- First$_1(L_1 \cup L_2) = \text{First}_1(L_1) \cup \text{First}_1(L_2)$
- First$_1(L_1 \cdot L_2) = \text{First}_1(\text{First}_1(L_1) \cdot \text{First}_1(L_2))$

$I$ = concatenation

**Observation:**
Let $L_1, L_2 \subseteq T \cup \{\epsilon\}$ with $L_1 \neq \emptyset \neq L_2$. Then:

$$L_1 \cdot L_2 = \begin{cases} L_1 \setminus \{\epsilon\} & \text{if } \epsilon \notin L_1 \\ (L_1 \setminus \{\epsilon\}) \cup L_2 & \text{otherwise} \end{cases}$$

If all rules of $G$ are productive, then all sets First$_1(A)$ are non-empty.

---

Lookahead Sets

For $\alpha \in (N \cup T)^*$ we are interested in the set:

$$\text{First}_1(\alpha) = \text{First}_1(\{w \in T^* \mid \alpha \rightarrow^* w\})$$

**Idea:** Treat $\epsilon$ separately: $F_\epsilon$

- Let $\text{empty}(X) = \text{true}$ iff $X \rightarrow^* \epsilon$.
- $F_\epsilon(X_1 \ldots X_m) = \bigcup_{i=1}^m F_\epsilon(X_i)$ if $\text{empty}(X_1) \wedge \ldots \wedge \text{empty}(X_{j-1})$.

We characterize the $\epsilon$-free First$_1$-sets with an inequality system:

- $F_\epsilon(\alpha) = \{\alpha\}$ if $\alpha \in T$
- $F_\epsilon(A) \supseteq \bigcup F_\epsilon(X_i)$ if $A \Rightarrow X_1 \ldots X_m \in P, \text{ empty}(X_1) \wedge \ldots \wedge \text{empty}(X_{j-1})$