Example:

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

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

Definition: Pushdown Automaton

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
... 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>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>
</tbody>
</table>

**Definition: Deterministic Pushdown Automaton**

The pushdown automaton $M$ is deterministic, if every configuration has maximally one successor configuration.

This is exactly the case if for distinct transitions $(\gamma_1, a, \gamma_2), (\gamma_3, a', \gamma_4) \in \delta$ we can assume:

- $\gamma_1$ a suffix of $\gamma_3'$, then $x \neq x' \land x \neq \epsilon \neq x'$ is valid.

---

**Pushdown Automata**

**Theorem:**

For each context free grammar $G = (N, T, P, S)$ a pushdown automaton $M$ with $L(G) = L(M)$ 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:

- $M_f$ to build Leftmost derivations
- $M_r$ to build reverse Rightmost derivations

---

**Syntactic Analysis**

**Chapter 3:**

Top-down Parsing
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:

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>New State</th>
</tr>
</thead>
<tbody>
<tr>
<td>( [S' \rightarrow S] )</td>
<td>( \varepsilon )</td>
<td>( [S' \rightarrow S \bullet] )</td>
</tr>
<tr>
<td>( [S \rightarrow \bullet S] )</td>
<td>( \varepsilon )</td>
<td>( [S \rightarrow \bullet AB] )</td>
</tr>
<tr>
<td>( [A \rightarrow \bullet a] )</td>
<td>( a )</td>
<td>( [A' \rightarrow \bullet a] )</td>
</tr>
<tr>
<td>( [S \rightarrow AB] )</td>
<td>( A \rightarrow a, B \rightarrow b )</td>
<td>( [S \rightarrow S \bullet] )</td>
</tr>
</tbody>
</table>

Item Pushdown Automaton

The item pushdown automaton \( M_L^I \) has three kinds of transitions:

**Expansions:** \( \{(A \rightarrow a \bullet B \beta) \in (A \rightarrow \alpha B \beta | B \rightarrow \gamma)\} \) for \( A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P \)

**Shifts:** \( \{(A \rightarrow \alpha a \beta) \in (A \rightarrow \alpha a \beta | A \rightarrow \alpha a)\} \) for \( A \rightarrow \alpha a \beta \in P \)

**Reduces:** \( \{(A \rightarrow \alpha B \beta | B \rightarrow \gamma, \epsilon, [A \rightarrow \alpha B \beta)]\} \) for \( A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P \)

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

The item pushdown automaton shifts the bullet 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 \( [A \rightarrow \alpha B \beta] \) the following holds:
  \( ([A \rightarrow \alpha B \beta], w) \rightarrow^+ ([A \rightarrow \alpha B \beta], \epsilon) \) if \( \& B \rightarrow^* \lambda \)
- LL-Parsing is based on the item pushdown automaton and tries to make the expansions deterministic ...

Item Pushdown Automaton

The item pushdown automaton \( M_L^I \) has three kinds of transitions:

**Expansions:** \( \{(A \rightarrow a B \beta) \in (A \rightarrow \alpha B \beta | B \rightarrow \gamma)\} \) for \( A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P \)

**Shifts:** \( \{(A \rightarrow \alpha a \beta) \in (A \rightarrow \alpha a \beta | A \rightarrow \alpha a)\} \) for \( A \rightarrow \alpha a \beta \in P \)

**Reduces:** \( \{(A \rightarrow \alpha B \beta | B \rightarrow \gamma, \epsilon, [A \rightarrow \alpha B \beta])\} \) for \( A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P \)

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

The item pushdown automaton shifts the bullet around the derivation tree ...
Item Pushdown Automaton

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], w) \rightarrow^* ([A \rightarrow \alpha B \bullet \beta], \epsilon) \iff B \rightarrow^* w

- LL-Parsing is based on the item pushdown automaton and tries to make the expansions deterministic ...

Example: \[ S \rightarrow c | a \; S \; b \]

The transitions of the according Item Pushdown Automaton:

<table>
<thead>
<tr>
<th>0</th>
<th>[ S' \rightarrow \bullet S ]</th>
<th>[ S' \rightarrow \bullet S ]</th>
<th>[ S' \rightarrow \bullet S ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[ S' \rightarrow S ]</td>
<td>[ S' \rightarrow S ]</td>
<td>[ S' \rightarrow S ]</td>
</tr>
<tr>
<td>2</td>
<td>[ S \rightarrow a ; S ; b ]</td>
<td>[ S \rightarrow a ; S ; b ]</td>
<td>[ S \rightarrow a ; S ; b ]</td>
</tr>
<tr>
<td>3</td>
<td>[ S \rightarrow a ; S ; b ]</td>
<td>[ S \rightarrow a ; S ; b ]</td>
<td>[ S \rightarrow a ; S ; b ]</td>
</tr>
<tr>
<td>4</td>
<td>[ S \rightarrow a ; S ; b ]</td>
<td>[ S \rightarrow a ; S ; b ]</td>
<td>[ S \rightarrow a ; S ; b ]</td>
</tr>
<tr>
<td>5</td>
<td>[ S \rightarrow a ; S ; b ]</td>
<td>[ S \rightarrow a ; S ; b ]</td>
<td>[ S \rightarrow a ; S ; b ]</td>
</tr>
<tr>
<td>6</td>
<td>[ S \rightarrow a ; S ; b ]</td>
<td>[ S \rightarrow a ; S ; b ]</td>
<td>[ S \rightarrow a ; S ; b ]</td>
</tr>
<tr>
<td>7</td>
<td>[ S \rightarrow a ; S ; b ]</td>
<td>[ S \rightarrow a ; S ; b ]</td>
<td>[ S \rightarrow a ; S ; b ]</td>
</tr>
<tr>
<td>8</td>
<td>[ S' \rightarrow \bullet S ]</td>
<td>[ S \rightarrow S ]</td>
<td>[ S' \rightarrow S ]</td>
</tr>
<tr>
<td>9</td>
<td>[ S' \rightarrow S ]</td>
<td>[ S \rightarrow a ; S ; b ]</td>
<td>[ S' \rightarrow S ]</td>
</tr>
</tbody>
</table>

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

Problem:

Conflicts between the transitions prohibit an implementation of the item pushdown automaton as deterministic pushdown automaton.
Topdown Parsing

Idea:
- Emanate from the item pushdown automaton
- Consider the next input 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 $S \rightarrow \alpha A \beta$ and $S \rightarrow \alpha' A \beta'$ with $u \in T^*$, the following is valid:
$$\text{First}_1(\alpha \beta) \cap \text{First}_1(\alpha' \beta') = \emptyset$$

Topdown Parsing

Example 1:
\[
\begin{align*}
S & \rightarrow \text{if } (E) S \text{ else } S \\
 & \quad \text{while } (E) S \\
E & \rightarrow \text{id}
\end{align*}
\]

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

Lookahead Sets

Definition: First$_1$-Sets
For a set $L \subseteq T^*$ we define:
$$\text{First}_1(L) = \{e \in T \mid \exists v \in T^* : e \in L\}$$

Example: $S \rightarrow \epsilon \mid a S b$

\[
\begin{array}{c}
\text{First}_1(S) \\
\{a\} \\
\{a, bb\} \\
\{a, a bb\} \\
\ldots
\end{array}
\]
Lookahead Sets

Arithmetics:
First(\emptyset) = \emptyset
First(L_1 \cup L_2) = First_1(L_1) \cup First_1(L_2)
First(L_1 \cdot L_2) = First_1(First_1(L_1) \cdot First_1(L_2))
implies First_1(L_1) \cdot First_1(L_2)

\(\emptyset\) being \(1\) – concatenation

Lookahead Sets

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

\[\text{First}_1(a) = \{w \in T^* \mid a \rightarrow^* w\}\]

Idea: Treat \(\epsilon\) separately:
\[\text{First}_1(A) = \text{First}_1(A) \cup \{A \rightarrow \epsilon\}\]

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

Fast Computation of Lookahead Sets

Observation:
- The form of each inequality of these systems is:
  \[x \lessdot y \text{ resp. } x \lessdot d\]
  for variables \(x, y\) and \(d \in D\).
- Such systems are called pure unification problems.
- Such problems can be solved in linear time.
  for example:

\[
\begin{align*}
x_0 &\geq (a) \\
x_1 &\geq (b) \\
x_2 &\geq (c) \\
x_3 &\geq (a) \\
x_4 &\geq (c)
\end{align*}
\]

\[
\begin{align*}
x_0 &\geq (a) \\
x_1 &\geq (b) \\
x_2 &\geq (c) \\
x_3 &\geq (a) \\
x_4 &\geq (c)
\end{align*}
\]

Define: \(1\)-concatenation

Let \(L_1, L_2 \subseteq T \cup \{\epsilon\}\) with \(L_1 \neq \emptyset \neq L_2\). Then:

\[L_1 \cdot L_2 = \begin{cases} 
L_1 & \text{if } \epsilon \notin L_1 \\
L_2 & \text{otherwise} 
\end{cases}
\]

If all rules of \(G\) are productive, then all sets \(\text{First}_1(A)\) are non-empty.
Fast Computation of Lookahead Sets

Proceeding:
- Create the Variable Dependency Graph for the inequality system.

Item Pushdown Automaton as LL(1)-Parser

back to the example: \( S \rightarrow \epsilon | a \ S \ b \)

The transitions in the according Item Pushdown Automaton:

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

Item Pushdown Automaton as LL(1)-Parser

... in detail: \( S \rightarrow \epsilon | a \ S \ b \)

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