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 $(\gamma_1, x, \gamma_2), (\gamma'_1, x', \gamma'_2) \in \delta$ we can assume:
Is $\gamma_1$ a suffix of $\gamma'_1$, then $x \neq x' \land x \neq \epsilon \neq x'$ is valid.

... for example:

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

... this obviously holds
**Item Pushdown Automaton**

**Construction:** Item Pushdown Automaton $M_G^L$

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

The states are now items $= \{ [A \rightarrow \alpha \bullet \beta] \}$...

The bullet 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**

---

**Our example:**

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

**Our example:**

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

![Diagram of an automaton](image)

Item Pushdown Automaton

The item pushdown automaton \( M_k \) has three kinds of transitions:

**Expansions:**

\[ ([A \rightarrow a \bullet B \beta], \epsilon, [A \rightarrow a \bullet B \beta], B \rightarrow \gamma) \]

for

\[ A \rightarrow a B \beta, B \rightarrow \gamma \in P \]

**Shifts:**

\[ ([A \rightarrow a \bullet a \beta], \epsilon, [A \rightarrow a \bullet \beta]) \]

for

\[ A \rightarrow a a \beta \in P \]

**Reduces:**

\[ ([A \rightarrow a \bullet B \beta], B \rightarrow \gamma, \epsilon, [A \rightarrow a B \bullet \beta]) \]

for

\[ A \rightarrow a B \beta, B \rightarrow \gamma \in P \]

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

The item pushdown automaton shifts the bullet around the derivation tree ...

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:**

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

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 \([A \rightarrow a \bullet B \beta] \) the following holds:

\[
([A \rightarrow a \bullet B \beta], w) \vdash^* ([A \rightarrow a B \bullet \beta], \epsilon) \quad \text{iff} \quad B \vdash^* w
\]

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

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

The transitions of the according Item Pushdown Automaton:

<table>
<thead>
<tr>
<th>#</th>
<th>Transition</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$S' \rightarrow \bullet S$</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>1</td>
<td>$S' \rightarrow \bullet S$</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>2</td>
<td>$S \rightarrow \bullet a S b$</td>
<td>$a$</td>
</tr>
<tr>
<td>3</td>
<td>$S \rightarrow a \bullet S b$</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>4</td>
<td>$S \rightarrow a \bullet S b$</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>5</td>
<td>$S \rightarrow a S b \rightarrow \bullet$</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>6</td>
<td>$S \rightarrow a S b \rightarrow a S b \bullet$</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>7</td>
<td>$S \rightarrow a S b \rightarrow b$</td>
<td>$S \rightarrow a S b \bullet$</td>
</tr>
<tr>
<td>8</td>
<td>$S' \rightarrow \bullet S \rightarrow \bullet$</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>9</td>
<td>$S' \rightarrow \bullet S \rightarrow a S b \bullet$</td>
<td>$\epsilon$</td>
</tr>
</tbody>
</table>

Topdown Parsing

Problem:

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

Idea 1: GLL Parsing

For each conflict, we create a virtual copy of the complete stack and continue deriving in parallel.

Idea 2: Recursive Descent & Backtracking

Depth-first search for an appropriate derivation.
Topdown Parsing

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

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

Idea 2: Recursive Descent & Backtracking
Depth-first search for an appropriate derivation.

Idea 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 $M[i, w]$ contains the rule of choice.

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 $A \rightarrow x \in P$, $A \rightarrow y \in P$ and each derivation $S \rightarrow^*_\gamma u$, with $u \in T^*$ the following is valid:

$$\text{First}_x(\gamma) \cap \text{First}_y(\gamma) = \emptyset$$
Topdown Parsing

Example 1:

\[ S \rightarrow \begin{cases} \text{if} & (E) S \text{ else } S \mid \\
\text{while} & (E) S \mid \\
E \end{cases} \]

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

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

Lookahead Sets

**Definition: First₁-Sets**

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

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

Example: \( S \rightarrow \epsilon \mid a \ S \ b \)

\[
\begin{array}{c|c|c}
\text{First}_1(S) \\
\hline
\epsilon & \epsilon, a \\
ab & a \\
a a b b & a a b b \\
a a a b b & \ldots \\
\end{array}
\]

\( \equiv \) the yield’s prefix of length \( 1 \)
Lookahead Sets

Arithmetics:
First(_X) is compatible with union and concatenation:

First(_X) = ∅
First(_X \cup \_Y) = First(_X) \cup First(_Y)
First(_X \cdot \_Y) = First(First(_X) \cdot First(_Y))

○ being 1 – concatenation

Lookahead Sets

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

First(_X) = First(_X)\{w ∈ T^* | α →^* w\}

Idea: Treat ε separately: First(_X) = F_ε(_X) \cup \{ε | \alpha \rightarrow^* ε\}

- Let empty(X) = true if X →^* ε
- F_ε(X_1 \ldots X_m) = \bigcup_{i=1}^{m} F_ε(X_i) if \bigwedge_{i=1}^{m} empty(X_i)

Lookahead Sets

Arithmetics:
First(_X) is compatible with union and concatenation:

First(_X) = ∅
First(_X \cup \_Y) = First(_X) \cup First(_Y)
First(_X \cdot \_Y) = First(First(_X) \cdot First(_Y))

○ being 1 – concatenation

Definition: 1-concatenation

Let L_1, L_2 ⊆ T \cup \{ε\} with L_1 \neq \emptyset \neq L_2. Then:

L_1 \odot L_2 = \begin{cases} L_1 \setminus \{ε\} \cup L_2 & \text{if } ε \notin L_1 \\ L_2 & \text{otherwise} \end{cases}

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

Lookahead Sets

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

First(_X) = First(_X)\{w ∈ T^* | α →^* \_X\}

Idea: Treat ε separately: First(_X) = F_ε(_X) \cup \{ε | \alpha \rightarrow^* \_X\}

- Let empty(X) = true if X →^* ε
- F_ε(X_1 \ldots X_m) = \bigcup_{i=1}^{m} F_ε(X_i) if \bigwedge_{i=1}^{m} empty(X_i)

We characterize the ε-free First(_X) sets with an inequality system:

\begin{align*}
F_ε(a) &= \{a\} \quad \text{if } a \in T \\
F_ε(A) &\geq F_ε(X_i) \quad \text{if } A \rightarrow X_1 \ldots X_m \in P, \\bigwedge_{i=1}^{m} empty(X_i)
\end{align*}
for example...

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid \text{name} \mid \text{int}$

with $\text{empty}(E) = \text{empty}(T) = \text{empty}(F) = \text{false}$

... we obtain:

$F_*(S') \supseteq F_*(E)$

$F_*(E) \supseteq F_*(F)$

$\{ \text{name}, \text{int} \}$

for example...

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid \text{name} \mid \text{int}$

with $\text{empty}(E) = \text{empty}(T) = \text{empty}(F) = \text{false}$

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: $\text{First}_1(A) = F_*(A) \cup \{ \epsilon \mid A \rightarrow^* \epsilon \}$

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

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

$F_*(a) = \{ a \}$ if $a \in T$

$F_*(A) \supseteq F_*(X)$ if $A \rightarrow X_1 \ldots X_m \in P$,

$\bigwedge_{i=1}^{m-1} \text{empty}(X_i)$
Fast Computation of Lookahead Sets

Observation:
- The form of each inequality of these systems is:
  \[ x \geq y \quad \text{resp.} \quad x \trianglerighteq d \]
  for variables \( x, y \) und \( d \in D \).
- Such systems are called pure unification problems.
- Such problems can be solved in linear space/time.
  for example:
  \[ D = \{a,b,c\} \]

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

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

---

Proceeding:
- Create the Variable Dependency Graph for the inequality system.
- Within a Strongly Connected Component (\( \rightarrow \) Tarjan) all variables have the same value.
- Is there no ingoing edge for an SCC, its value is computed via the smallest upper bound of all values within the SCC.
Fast Computation of Lookahead Sets

**Proceeding:**
- Create the Variable Dependency Graph for the inequality system.
- Within a Strongly Connected Component (→ Tarjan) all variables have the same value.
- Is there no ingoing edge for an SCC, its value is computed via the smallest upper bound of all values within the SCC.

---

Fast Computation of Lookahead Sets

... for our example grammar:

First₁:

```
S' → E T
E → , int, name
T → , int, name
```

---

Item Pushdown Automaton as LL(1)-Parser

**back to the example:** \( S \rightarrow e \mid a S b \)

The transitions in 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 \bullet S )</td>
<td>( e )</td>
</tr>
<tr>
<td>1</td>
<td>( S' \rightarrow \bullet S )</td>
<td>( e )</td>
</tr>
<tr>
<td>2</td>
<td>( S \rightarrow a S b )</td>
<td>( a )</td>
</tr>
<tr>
<td>3</td>
<td>( S \rightarrow a S b )</td>
<td>( e )</td>
</tr>
<tr>
<td>4</td>
<td>( S \rightarrow a S b )</td>
<td>( e )</td>
</tr>
<tr>
<td>5</td>
<td>( S \rightarrow a S b )</td>
<td>( e )</td>
</tr>
<tr>
<td>6</td>
<td>( S \rightarrow a S b )</td>
<td>( e )</td>
</tr>
<tr>
<td>7</td>
<td>( S \rightarrow a S b )</td>
<td>( b )</td>
</tr>
<tr>
<td>8</td>
<td>( S' \rightarrow \bullet S )</td>
<td>( e )</td>
</tr>
<tr>
<td>9</td>
<td>( S' \rightarrow \bullet S )</td>
<td>( e )</td>
</tr>
</tbody>
</table>

Conflicts arise between transitions \((0, 1)\) or \((3, 4)\) resp.
Item Pushdown Automaton as LL(1)-Parser

... in detail:  \[ S \rightarrow e^0 \mid a \, S \, b^1 \]

<table>
<thead>
<tr>
<th>First$_1$ (input)</th>
<th>e</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

Inequality system for Follow$_1(B) = \text{First}$_1(\beta) \circ \ldots \circ \text{First}$_1(\beta_0)

\[
\begin{align*}
\text{Follow}_1(S) & \supseteq \{\epsilon\} \\
\text{Follow}_1(B) & \supseteq F_e(X_1) \quad \text{if} \quad A \rightarrow \alpha \, B \, X_1 \ldots \, X_m \in P, \\
& \phantom{=} \text{empty}(X_1) \land \ldots \land \text{empty}(X_{j-1}) \\
& \phantom{=} \land \text{empty}(X_j) \land \ldots \land \text{empty}(X_m) \\
\text{Follow}_1(A) & \supseteq \text{Follow}_1(A) \quad \text{if} 
\end{align*}
\]
Item Pushdown Automaton as LL(1)-Parser

Is \( G \) an LL(1)-grammar, we can index a lookahead-table with items and nonterminals:

**LL(1)-Lookahead Table**

We set \( M[B, w] = i \) with \( B \rightarrow \gamma^i \) exactly if

- \( S' \rightarrow \gamma^i u B \beta \)
- \( w \in \text{First}_1(\gamma) \odot \text{Follow}_1(\beta) \)

... for example: \( S \rightarrow \epsilon^0 \mid a S b^1 \)

**First_1(S) = \{\epsilon, a\}**

**Follow_1(S) = \{b, \epsilon\}**
Item Pushdown Automaton as LL(1)-Parser

Is \( G \) an LL(1)-grammar, we can index a lookahead-table with items and nonterminals:

**LL(1)-Lookahead Table**

We set \( M[B, w] = i \) with \( B \rightarrow \gamma \) exactly if

- \( S' \rightarrow_1 u B \beta \)
- \( w \in \text{First}_1(\gamma) \cap \text{Follow}_1(\beta) \)

... for example:

\[
S \rightarrow \epsilon^1 \mid a S b^1
\]

\( \text{First}_1(S) = \{ \epsilon, a \} \quad \text{Follow}_1(S) = \{ b, \epsilon \} \)

\( S \)-rule 0:

\[
\begin{array}{c|c|c}
\text{First}_1(\epsilon) & \cap & \text{Follow}_1(S) = \{ b, \epsilon \} \\
\hline
S & 0 & 1 \end{array}
\]

\( S \)-rule 1:

\[
\begin{array}{c|c|c}
\text{First}_1(a S b) & \cap & \text{Follow}_1(S) = \{ a \} \\
\hline
S' & 1 & 0 \end{array}
\]

For example:

\[
S \rightarrow \epsilon^1 \mid a S b^1
\]

The transitions of the according Item Pushdown Automaton:

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( S' \rightarrow \epsilon )</td>
<td>( \epsilon )</td>
</tr>
<tr>
<td>1</td>
<td>( S' \rightarrow S )</td>
<td>( \epsilon )</td>
</tr>
<tr>
<td>2</td>
<td>( S \rightarrow a S b )</td>
<td>( a )</td>
</tr>
<tr>
<td>3</td>
<td>( S \rightarrow a )</td>
<td>( \epsilon )</td>
</tr>
<tr>
<td>4</td>
<td>( S \rightarrow a b )</td>
<td>( \epsilon )</td>
</tr>
<tr>
<td>5</td>
<td>( S \rightarrow a S b )</td>
<td>( b )</td>
</tr>
<tr>
<td>6</td>
<td>( S \rightarrow a S b )</td>
<td>( \epsilon )</td>
</tr>
<tr>
<td>7</td>
<td>( S \rightarrow a S b )</td>
<td>( \epsilon )</td>
</tr>
<tr>
<td>8</td>
<td>( S' \rightarrow S )</td>
<td>( \epsilon )</td>
</tr>
<tr>
<td>9</td>
<td>( S' \rightarrow S )</td>
<td>( \epsilon )</td>
</tr>
</tbody>
</table>

Lookahead table:

<table>
<thead>
<tr>
<th>Lookahead</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \epsilon )</td>
<td>1</td>
</tr>
<tr>
<td>( a )</td>
<td>1</td>
</tr>
<tr>
<td>( b )</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \epsilon )</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>( S )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( a )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( b )</td>
<td></td>
</tr>
</tbody>
</table>

**Discussion**

- A practical implementation of an LL(1)-parser via recursive Descent is a straight-forward idea.
- However, only a subset of the deterministic contextfree languages can be parsed this way.

**Discussion**

- A practical implementation of an LL(1)-parser via recursive Descent is a straight-forward idea.
- However, only a subset of the deterministic contextfree languages can be parsed this way.
- The size of the occurring sets is rapidly increasing with larger \( k \).
- Unfortunately, even \( LL(k) \) parsers are not sufficient to accept all deterministic contextfree languages.
- In practical systems, this often motivates the implementation of \( k = 1 \) only...
Chapter 4:
Bottom-up Analysis