Syntactic Analysis

- Syntax analysis tries to integrate Tokens into larger program units.

Item Pushdown Automaton – Example

Our example:

- $S \rightarrow AB$
- $A \rightarrow a$
- $B \rightarrow b$
Our example:

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

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

- Start state: \( S' \rightarrow S \)
- End state: \( S \rightarrow S \bullet \)

Transition relations:

\[
\begin{array}{c|c}
S' & S \rightarrow S \\
S & S \rightarrow \bullet S \\
A & A \rightarrow a \\
S & S \rightarrow A B \\
B & B \rightarrow b \\
S' & S \rightarrow A B \\
\end{array}
\]

\[
\begin{array}{c|c}
S' \rightarrow S & \epsilon \\
S \rightarrow S & \epsilon \\
S \rightarrow A B & \epsilon \\
S \rightarrow A \bullet B & \epsilon \\
S \rightarrow A B & \epsilon \\
\end{array}
\]

\[
\begin{array}{c|c}
A \rightarrow a & A \rightarrow a \bullet \\
S \rightarrow A B & S \rightarrow A B \\
S \rightarrow A \bullet B & S \rightarrow A \bullet B \\
\end{array}
\]

The expanded transitions are shown in the table above.

Item Pushdown Automaton

The item pushdown automaton \( M_G^I \) 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: \([A \rightarrow a \bullet]\) are also called complete.

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

Item Pushdown Automaton

The item pushdown automaton \( M_G^I \) 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: \([A \rightarrow a \bullet]\) 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:

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

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

**Beispiel:** \( S \rightarrow e \mid a S b \)

The transitions of the according Item Pushdown Automaton:

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

---

**Topdown Parsing**

**Problem:**

Conflicts between the transitions prohibit an implementation of the item pushdown automaton as deterministic pushdown automaton.
**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 $M$ 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 $A \rightarrow \alpha$ and $A \rightarrow \alpha'$ with $A \in T^*$ and each derivation $S \rightarrow^*_L AA\beta$ the following is valid:

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

Example 1:

\[
S \rightarrow \text{if } (E) S \text{ else } S \\
\text{while } (E) S \\
E: \\
E \rightarrow \text{id}
\]

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

Example 2:

\[
S \rightarrow \text{if } (E) S \text{ else } S \\
\text{if } (E) S \\
\text{while } (E) S \\
E: \\
E \rightarrow \text{id}
\]

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

Lookahead Sets

Definition:

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

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

Example:

\begin{align*}
S & \rightarrow \varepsilon \\
S & \rightarrow aSb \\
\text{First}_n(S) & = \{e, ab, aabb, aaabb, aabbb\}
\end{align*}

the prefixes of length 1
Lookahead Sets

Arithmetics:
First ( ) is compatible with union and concatenation:
\[
\begin{align*}
\text{First}_i(\emptyset) &= \emptyset \\
\text{First}_i(L_1 \cup L_2) &= \text{First}_i(L_1) \cup \text{First}_i(L_2) \\
\text{First}_i(L_1 \cdot L_2) &= \text{First}_i(\text{First}_i(L_1) \cdot \text{First}_i(L_2)) \\
&= \text{First}_i(L_1) \odot \text{First}_i(L_2)
\end{align*}
\]

I = concatenation

Observation:
Let \( L_1, L_2 \subseteq T \cup \{\epsilon\} \) with \( L_1 \neq \emptyset \neq L_2 \). Then:
\[
L_1 \odot L_2 = \begin{cases} 
(L_1 \setminus \{\epsilon\}) \cup L_2 & \text{if } \epsilon \notin L_1 \\
\emptyset & \text{otherwise}
\end{cases}
\]

If all rules of \( G \) are productive, then all sets \( \text{First}_i(A) \) are non-empty.

Lookahead Sets

For \( \alpha \in (N \cup T)^* \) we are interested in the set:
\[
\text{First}_i(\alpha) = \text{First}_i(\{w \in T^* \mid \alpha \rightarrow^* w\})
\]

Idea: Treat \( \epsilon \) separately: \( F_\epsilon \)
\( \cdot \) Let \( \text{empty}(X) = \text{true} \) iff \( X \rightarrow^* \epsilon \).
\( \cdot \) \( F_\epsilon(X_1 \ldots X_m) = \bigcup_{j=1}^m F_\epsilon(X_i) \) if \( \text{empty}(X_1) \land \ldots \land \text{empty}(X_{j-1}) \)

We characterize the \( \epsilon \)-free \( \text{First}_i \)-sets with an inequality system:
\[
\begin{align*}
F_\epsilon(a) &= \{a\} & \text{if } a \in T \\
F_\epsilon(A) &\supseteq F_\epsilon(X_j) & \text{if } \begin{align*} 
A &\rightarrow X_1 \ldots X_m \in P, \\
\text{empty}(X_1) \land \ldots \land \text{empty}(X_{j-1})
\end{align*}
\]

Lookahead Sets

for example...

\[
\begin{align*}
E &\rightarrow E + T & | & T \\
T &\rightarrow T + F & | & F \\
F &\rightarrow \{E\} & | & \text{name} \lor \text{int}
\end{align*}
\]

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

for example...

\[
E \rightarrow E + T \quad | \quad T \\
T \rightarrow T * F \quad | \quad F \\
F \rightarrow (E) \quad | \quad \text{name} \quad | \quad \text{int}
\]

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

... we obtain:

\[
\begin{align*}
F_s(S') & \cup \quad F_s(E) \quad F_s(E) \\
F_s(E) & \cup \quad F_s(T) \quad F_s(T) \\
F_s(T) & \cup \quad F_s(F) \quad F_s(F) \\
\end{align*}
\]

\{ , name, int \}

Fast Computation of Lookahead Sets

Observation:

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

for example:

\[ D = 2^{\{a,b,c\}} \]

Fast Computation of Lookahead Sets

Proceeding:

- Create the Variable dependency graph for the inequality system.

Fast Computation of Lookahead Sets

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 incoming edge for an SCC, its value is computed via the smallest upper bound of all values within the SCC.
- In case of incoming edges, their values are also to be considered for the upper bound.

Item Pushdown Automaton as LL(1)-Parser

back to the example: $S \rightarrow \epsilon | a\cdot S\cdot b$

The transitions in the according Item Pushdown Automaton:

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

Fast Computation of Lookahead Sets

... for our example grammar:

First$_1$:

Item Pushdown Automaton as LL(1)-Parser

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

We set $M[B, w] = i$ exactly if $(B, i)$ is the rule $B \rightarrow \gamma$ and:

$w \in \text{First}_1(\gamma) \cup \{ \text{First}_1(\beta) | S \rightarrow \gamma \cdot u \cdot B \cdot \beta \}.$

... for example:

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

back to the example: \( S \to \varepsilon \mid aSb \)

The transitions in the according Item Pushdown Automaton:

<table>
<thead>
<tr>
<th>( i )</th>
<th>( S \to \bullet S )</th>
<th>( \varepsilon )</th>
<th>( S \to \bullet S )</th>
<th>( S \to \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( S \to \bullet S )</td>
<td>( \varepsilon )</td>
<td>( S \to \bullet S )</td>
<td>( S \to \varepsilon )</td>
</tr>
<tr>
<td>1</td>
<td>( S \to \bullet S )</td>
<td>( \varepsilon )</td>
<td>( S \to \bullet S )</td>
<td>( S \to \varepsilon )</td>
</tr>
<tr>
<td>2</td>
<td>( S \to a S )</td>
<td>( a )</td>
<td>( S \to a S )</td>
<td>( S \to a \varepsilon )</td>
</tr>
<tr>
<td>3</td>
<td>( S \to a S )</td>
<td>( \varepsilon )</td>
<td>( S \to a S )</td>
<td>( S \to \varepsilon )</td>
</tr>
<tr>
<td>4</td>
<td>( S \to a S )</td>
<td>( \varepsilon )</td>
<td>( S \to a S )</td>
<td>( S \to \varepsilon )</td>
</tr>
<tr>
<td>5</td>
<td>( S \to a S )</td>
<td>( \varepsilon )</td>
<td>( S \to a S )</td>
<td>( S \to \varepsilon )</td>
</tr>
<tr>
<td>6</td>
<td>( S \to a S )</td>
<td>( \varepsilon )</td>
<td>( S \to a S )</td>
<td>( S \to \varepsilon )</td>
</tr>
<tr>
<td>7</td>
<td>( S \to a S )</td>
<td>( \varepsilon )</td>
<td>( S \to a S )</td>
<td>( S \to \varepsilon )</td>
</tr>
<tr>
<td>8</td>
<td>( S \to \varepsilon )</td>
<td>( \varepsilon )</td>
<td>( S \to \varepsilon )</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>( S \to a S )</td>
<td>( \varepsilon )</td>
<td>( S \to a S )</td>
<td>( S \to \varepsilon )</td>
</tr>
</tbody>
</table>

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

Item Pushdown Automaton as LL(1)-Parser

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

We set \( M[B, w] = i \) exactly if \( (B, i) \) is the rule \( B \to \gamma \) and:

\[ w \in \text{First}_i(\gamma) \cup \{ \text{First}_i(\beta) | S \to \varepsilon \cup B \beta \} \]

... for example:

\[ S \to \varepsilon \mid aSb \]

\[ w \in \text{First}_i(\gamma) \]

Inequality system for \( \text{Follow}_i(B) = \{ \text{First}_i(\beta) | S \to \varepsilon \cup B \beta \} \)

\[ \text{Follow}_i(B) \supseteq \{ \text{Follow}_i(A) \text{ if } \forall A \to \alpha B X_1 \ldots X_m \in P, \text{ empty}(X_1) \land \ldots \land \text{ empty}(X_m) \} \]

Lookahead table:
**Item Pushdown Automaton as LL(1)-Parser**

Inequality system for $\text{Follow}_1(B) = \bigcup \{ \text{First}_1(\beta) \mid S' \rightarrow^*_L \alpha B \beta \}$

- $\text{Follow}_1(S) \supseteq \{ \epsilon \}$
- $\text{Follow}_1(B) \supseteq F_i(X_j)$ if $A \rightarrow \alpha BX_1 \ldots X_m \in P,
  \text{empty}(X_1) \land \ldots \land \text{empty}(X_{j-1})$
- $\text{Follow}_1(B) \supseteq \text{Follow}_1(A)$ if $A \rightarrow \alpha BX_1 \ldots X_m \in P,
  \text{empty}(X_1) \land \ldots \land \text{empty}(X_m)$

**Topdown-Parsing**

**Discussion**

- A practical implementation of an $LL(1)$-parser via recursive Descent is a straightforward idea.
- However, only a subset of the deterministic contextfree languages can be read this way.
- **Solution:** Going from $LL(1)$ to $LL(k)$
- The size of the occuring 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 ...