Derivation Tree

Derivations of a symbol are represented as derivation trees:

... for example:

\[
E \quad \rightarrow \quad \begin{array}{l}
\rightarrow^0 \quad E + T \\
\rightarrow^1 \quad T + T \\
\rightarrow^2 \quad T \times F + T \\
\rightarrow^3 \quad E \times \text{int} + T \\
\rightarrow^4 \quad \text{name} \times \text{int} + T \\
\rightarrow^5 \quad \text{name} \times \text{int} + \text{int}
\end{array}
\]

A derivation tree for \( A \in \mathcal{N} \):
- inner nodes: rule applications
- root: rule application for \( A \)
- leaves: terminals or \( \epsilon \)

The successors of \((B, i)\) correspond to right hand sides of the rule

Special Derivations

Attention:
In contrast to arbitrary derivations, we find special ones, always rewriting the leftmost (or rather rightmost) occurrence of a nonterminal.

- These are called leftmost (or rather rightmost) derivations and are denoted with the index \( L \) (or \( R \) respectively).
- Leftmost (or rightmost) derivations correspond to a left-to-right (or right-to-left) preorder-DFS-traversal of the derivation tree.
- Reverse rightmost derivations correspond to a left-to-right postorder-DFS-traversal of the derivation tree.
Special Derivations

... for example:

Leftmost derivation: \[(E, 0) (E, 1) (T, 0) (T, 1) (F, 0) (F, 2) (T, 1) (F, 2) (E, 0) (T, 1) (F, 0) (F, 2) (T, 1) (F, 1)\]
Rightmost derivation: \[(E, 0) (E, 1) (T, 0) (T, 1) (F, 1) (F, 2) (T, 1) (F, 2) (E, 0) (T, 1) (F, 0) (F, 2) (T, 1) (F, 1)\]
Reverse rightmost derivation: \[(F, 1) (T, 1) (F, 2) (T, 0) (E, 1) (F, 2) (T, 1) (E, 0)\]

Unique Grammars

The concatenation of leaves of a derivation tree \( t \) are often called \( \text{yield}(t) \).

... for example:

\[
\begin{align*}
\text{E} & \rightarrow \text{E} + \text{E}^0 | \text{E} \times \text{E}^1 | (\text{E})^2 | \text{name}^3 | \text{int}^4 \\
\text{E} & \rightarrow \text{E} + \text{E}^0 |
\text{T} & \rightarrow \text{T} + \text{T}^0 | \text{T}^1 \\
\text{T} & \rightarrow \text{F}^0 | \text{F}^1 \\
\text{F} & \rightarrow (\text{E})^0 | \text{name}^1 | \text{int}^2
\end{align*}
\]

The first one is ambiguous, the second one is unique.
Conclusion:

- A derivation tree represents a possible hierarchical structure of a word.
- For programming languages, only those grammars with a unique structure are of interest.
- Derivation trees are one-to-one corresponding with leftmost derivations as well as (reverse) rightmost derivations.

Syntactic Analysis

Chapter 2:
Basics of Pushdown Automata

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.

---

**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
- $\delta \subseteq Q^+ \times (T \cup \{\epsilon\}) \times Q^*$ 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:

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

consisting of the pushdown content and the remaining input.
... for example:

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

0 \[\text{aabbbb}\] \rightarrow (11, \text{aabbbb})

(0, \text{aabbbb}) \rightarrow (11, \text{aabbbb})

... for example:

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

0 \[\text{aabbbb}\] \rightarrow (11, \text{aabbbb})

(0, \text{aabbbb}) \rightarrow (11, \text{aabbbb})
A computation step is characterized by the relation \( \vdash \subseteq (Q^* \times T^*)^2 \) with

\[
(\alpha \gamma, x_0 w) \vdash (\alpha \gamma', w) \quad \text{for} \quad (\gamma, x, \gamma') \in \delta
\]

**Remarks:**

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

\[
\mathcal{L}(M) = \{ w \in T^* \mid \exists f \in F : q_0 \xrightarrow{\gamma} \leftarrow f(w) \}
\]
**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 $x \neq x' \land x \neq \varepsilon \neq x'$ is valid.

... for example:

```
0  a  11
1  a  11
11 b  2
12 b  2
```

... this obviously holds

---

**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:

- $M^L$ to build **Leftmost derivations**
- $M^{R}$ to build **reverse Rightmost derivations**
Item Pushdown Automaton

Construction: Item Pushdown Automaton $M_L$

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

$\Rightarrow$ The states are now items (rules with a bullet):

$$[A \rightarrow \alpha \bullet \beta], \quad A \rightarrow \alpha \beta \in P$$

The bullet marks the spot, how far the rule is already processed.

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:

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

Item Pushdown Automaton

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

**Expansions:**

\[ ([A \rightarrow \alpha \cdot B \beta], \varepsilon, [A \rightarrow \alpha \cdot B \beta] [B \rightarrow \gamma]) \quad \text{for} \quad A \rightarrow \alpha B \beta, \quad B \rightarrow \gamma \in \mathcal{P} \]

**Shifting:**

\[ ([A \rightarrow \alpha \cdot B \beta], [A \rightarrow \alpha \cdot B \beta]) \quad \text{for} \quad A \rightarrow \alpha a \beta \in \mathcal{P} \]

**Reducing:**

\[ (A \rightarrow \alpha \cdot B \beta \beta, [A \rightarrow \alpha \cdot B \beta \beta]) \quad \varepsilon, \quad [A \rightarrow \alpha B \cdot \beta] \quad \text{for} \quad A \rightarrow \alpha B \beta, \quad B \rightarrow \gamma \in \mathcal{P} \]

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

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

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:**

\[
\begin{array}{c|c}
[S' \rightarrow \bullet S] & \varepsilon \quad [S' \rightarrow \bullet S] \\
[S \rightarrow \bullet AB] & \varepsilon \quad [S \rightarrow \bullet AB] [A \rightarrow \bullet a] \\
A \rightarrow \bullet a & \varepsilon \quad [A \rightarrow \bullet a] \\
[S \rightarrow \bullet AB] A \rightarrow \bullet a & \varepsilon \quad [S \rightarrow \bullet AB] [A \rightarrow \bullet a] \\
[S \rightarrow \bullet AB] & \varepsilon \quad [S \rightarrow \bullet AB] [B \rightarrow \bullet b] \\
[B \rightarrow \bullet b] & \varepsilon \quad [B \rightarrow \bullet b] \\
[S \rightarrow \bullet AB] [B \rightarrow \bullet b] & \varepsilon \quad [S \rightarrow \bullet AB] [B \rightarrow \bullet b] \\
[S' \rightarrow \bullet S] & \varepsilon \quad [S' \rightarrow \bullet S] \\
\end{array}
\]
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 $[A \rightarrow \alpha \bullet B \beta]$, the following holds:

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

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

Example:

$S \rightarrow \epsilon$ | $a S b$

The transitions of the according Item Pushdown Automaton:

<table>
<thead>
<tr>
<th>0</th>
<th>$S' \rightarrow S$</th>
<th>$\epsilon$</th>
<th>$S' \rightarrow S$</th>
<th>$S \rightarrow \epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$S' \rightarrow S$</td>
<td>$\epsilon$</td>
<td>$S' \rightarrow S$</td>
<td>$S \rightarrow a S b$</td>
</tr>
<tr>
<td>2</td>
<td>$S \rightarrow a S b$</td>
<td>$a$</td>
<td>$S \rightarrow a$</td>
<td>$S b$</td>
</tr>
<tr>
<td>3</td>
<td>$S \rightarrow a S b$</td>
<td>$\epsilon$</td>
<td>$S \rightarrow a S b$</td>
<td>$S \rightarrow \epsilon$</td>
</tr>
<tr>
<td>4</td>
<td>$S \rightarrow a S b$</td>
<td>$\epsilon$</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>$S \rightarrow \epsilon$</td>
<td>$S \rightarrow a S b$</td>
<td>$b$</td>
</tr>
<tr>
<td>6</td>
<td>$S \rightarrow a S b$</td>
<td>$S \rightarrow a S b$</td>
<td>$\epsilon$</td>
<td>$S \rightarrow a S b$</td>
</tr>
<tr>
<td>7</td>
<td>$S \rightarrow a S b$</td>
<td>$b$</td>
<td>$S \rightarrow a S b$</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>8</td>
<td>$S' \rightarrow S$</td>
<td>$S \rightarrow \epsilon$</td>
<td>$S' \rightarrow S$</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>$S' \rightarrow S$</td>
<td>$S \rightarrow a S b$</td>
<td>$\epsilon$</td>
<td>$S' \rightarrow S$</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.
**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 $M[q, w]$ contains the rule of choice.