... further examples:

\[ S \rightarrow \langle \text{stmt} \rangle \]
\[ \langle \text{stmt} \rangle \rightarrow \langle \text{if} \rangle \mid \langle \text{while} \rangle \mid \langle \text{exp} \rangle \]
\[ \langle \text{if} \rangle \rightarrow \text{if} \langle \text{exp} \rangle \langle \text{stmt} \rangle \text{else} \langle \text{stmt} \rangle \]
\[ \langle \text{while} \rangle \rightarrow \text{while} \langle \text{exp} \rangle \langle \text{stmt} \rangle \]
\[ \langle \text{exp} \rangle \rightarrow \text{int} \mid \langle \text{exp} \rangle \mid \langle \text{exp} \rangle = \langle \text{exp} \rangle \mid ... \]

Further conventions:

- For every nonterminal, we collect the right hand sides of rules and list them together.
- The \( j \)-th rule for \( A \) can be identified via the pair \( (A, j) \) with \( j \geq 0 \).

further grammars:

\[
\begin{align*}
E & \rightarrow E + E \mid E \times E \mid (E) \mid \text{name} \mid \text{int} \\
T & \rightarrow T + T \mid T \times T \\
F & \rightarrow (E) \mid \text{name} \mid \text{int}
\end{align*}
\]

Both grammars describe the same language
Derivation

Grammars are term rewriting systems. The rules offer feasible rewriting steps. A sequence of such rewriting steps $\alpha_0 \rightarrow \ldots \rightarrow \alpha_n$ is called derivation.

... for example: $E$ 

$E \rightarrow E + T$

$E \rightarrow T + T$

Derivation

Grammars are term rewriting systems. The rules offer feasible rewriting steps. A sequence of such rewriting steps $\alpha_0 \rightarrow \ldots \rightarrow \alpha_n$ is called derivation.

... for example: $E \rightarrow E + T$

$E \rightarrow T + T$

$T \rightarrow T * E + T$

$T \rightarrow T * \text{Int} + T$
Derivation

Grammars are term rewriting systems. The rules offer feasible rewriting steps. A sequence of such rewriting steps $\alpha_0 \rightarrow \ldots \rightarrow \alpha_m$ is called derivation.

... for example:

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

Definition

A derivation $\rightarrow$ is a relation on words over $N \cup T$, with

\[\alpha \rightarrow \alpha' \text{ iff } \alpha = \alpha_1 A \alpha_2 \land \alpha' = \alpha_1 \beta \alpha_2 \text{ for an } A \rightarrow \beta \in P \]

The reflexive and transitive closure of $\rightarrow$ is denoted as: $\rightarrow^*$
**Derivation**

Remarks:
- The relation $\rightarrow$ depends on the grammar.
- In each step of a derivation, we may choose:
  - a spot, determining where we will rewrite.
  - a rule, determining how we will rewrite.
- The language, specified by $G$, is:
  \[
  \mathcal{L}(G) = \{ w \in T^* \mid S \rightarrow^* w \}
  \]

Attention:
The order in which disjunct fragments are rewritten is not relevant.

**Special Derivations**

... for example:

```
E 0
  +
  E 1
    *
    T 0
      *
      F 1
        int
      F 2
        int
    F 2
```

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 preorder-DFS-traversal of the derivation tree.

```
Leftmost derivation: (E, 0) (E, 1) (T, 0) (F, 1) (F, 2) (T, 1) (F, 2) (E, 1) (T, 0) (F, 2) (T, 1) (F, 1)
Rightmost derivation: (E, 0) (T, 1) (F, 2) (E, 1) (T, 0) (F, 2) (T, 1) (F, 1)
```
**Special Derivations**

... for example:

```
E 0
  E 1 + T 1
  T 0 F 2
  T 1 * F 2 int
  F 1 name
```

Leftmost derivation: \((E, 0) (E, 1) (T, 0) (T, 1) (F, 1) (F, 2) (T, 1) (F, 2) (T, 1) (E, 0)\)

Rightmost derivation: \((E, 0) (T, 1) (F, 2) (E, 1) (T, 0) (F, 2) (T, 1) (F, 1)\)

Reverse rightmost derivation: \((F, 1) (T, 1) (F, 2) (T, 0) (E, 1) (F, 2) (T, 1) (E, 0)\)

gives rise to the concatenation: \(\text{name} \ast \text{int} + \text{int}\).

**Unique grammars**

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

... for example:

```
E 0
  E 1 + T 1
  T 0 F 2
  T 1 * F 2 int
  F 1 name
```

**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.
- Leftmost derivations correspond to a top-down reconstruction of the syntax tree.
- Reverse rightmost derivations correspond to a bottom-up reconstruction of the syntax tree.
Chapter 2: 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:

<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>
</tbody>
</table>

<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>
</tbody>
</table>

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 $M = (Q, \Sigma, \delta, q_0, F)$ with:
- $Q$ a finite set of states;
- $\Sigma$ an input alphabet;
- $q_0 \in Q$ the start state;
- $F \subseteq Q$ the set of final states and
- $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q^*$ 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:

$$\langle \cdot \rangle \quad w \in Q \times \Sigma^*$$

consisting of the pushdown content and the remaining input.

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

States:
- 0: $\delta(0, a, 0) := (1, a, 11)$
- 1: $\delta(1, a, 1) := (1, a, 11)$
- 11: $\delta(11, b, 2) := (12, b, 2)$
- 12: $\delta(12, b, 2) := (12, b, 2)$
... for example:

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

\[(0, \text{aabbbb}) \vdash (11, \text{aabbbb}) \vdash (111, \text{abb}) \vdash (1111, \text{b}) \]

... for example:

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

\[(0, \text{aabbbb}) \vdash (11, \text{aabbbb}) \vdash (111, \text{abb}) \vdash (1111, \text{b}) \]
A computation step is characterized by the relation
\[ \vdash \subseteq (Q^* \times T^*)^2 \]
with
\[ (\alpha \gamma, x w) \vdash (\alpha \gamma', w) \quad \text{for} \quad (\gamma, x, \gamma') \in \delta \]

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

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

- $M_G^L$ to build **Leftmost derivations**
- $M_G^R$ to build **reverse Rightmost derivations**

Chapter 3:
Top-down Parsing