Preliminary content

- Regular expressions and finite automata
- Specification and implementation of scanners
- Reduced context free grammars and pushdown automata
- Top-Down/Bottom-Up syntax analysis
- Attribute systems
- Type checking
- Code generation for register machines
- Register assignment
- Optional: Basic optimization

Extremes of Program Execution

Interpretation:
- Program
  - Input
  - Interpreter
  - Output

Compilation:
- Program
  - Input
  - Compiler
  - Code
  - Machine
  - Output

Interpretation vs. Compilation

Interpretation
- No precomputation on program text necessary
  - No/small startup overhead
- More context information allows for specific aggressive optimization

Compilation
- Program components are analyzed once, during preprocessing, instead of multiple times during execution
  - Smaller runtime overhead
- Runtime complexity of optimizations less important than in interpreter

Compiler

general Compiler setup:

- Program code
- Analysis
- Synthesis
- Compiler
- Code
The Compiler

The Analysis-Phase consists of several subcomponents:

The Lexical Analysis

Classified tokens allow for further pre-processing:

- **Dropping** irrelevant fragments e.g. Spacing, Comments, ...
- **Collecting Pragmas**, i.e. directives for the compiler, often implementation dependent, directed at the code generation process, e.g. OpenMP-Statements;
- **Replacing** of Tokens of particular classes with their meaning / internal representation, e.g.
  - Constants;
  - Names: typically managed centrally in a Symbol-table, maybe compared to reserved terms (if not already done by the scanner) and possibly replaced with an index or internal format (⇒ Name Mangling).

⇒ Siever

The Lexical Analysis

Discussion:

- Scanner and Siever are often combined into a single component, mostly by providing appropriate callback actions in the event that the scanner detects a token.
- Scanners are mostly not written manually, but generated from a specification.

The Lexical Analysis - Generating:

... in our case:

Specification → **Generator** → Scanner
Chapter 1: Basics: Regular Expressions

Regular Expressions

... Example:

\[(a \cdot b^* \cdot a)\]
\[(a \mid b)\]
\[((a \cdot b) \cdot (a \cdot b))\]

Regular Expressions

Specification needs Semantics

... Example:

<table>
<thead>
<tr>
<th>Specification</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>abab</td>
<td>{abab}</td>
</tr>
<tr>
<td>a \mid b</td>
<td>{a, b}</td>
</tr>
<tr>
<td>ab^*a</td>
<td>{ab^n</td>
</tr>
</tbody>
</table>

For \( e \in \mathcal{E}_\Sigma \) we define the specified language \([e] \subseteq \Sigma^*\) inductively by:

\[
[e] = \{e\}
\]
\[
[a] = \{a\}
\]
\[
[e^*] = ([e])^*
\]
\[
[e_1 \mid e_2] = [e_1] \cup [e_2]
\]
\[
[e_1 \cdot e_2] = [e_1] \cdot [e_2]
\]