Chapter 5: Scanner design

Scanner design

Input (simplified): a set of rules:

\[ e_1 \rightarrow \{ \text{action}_1 \} \]
\[ e_2 \rightarrow \{ \text{action}_2 \} \]
\[ \ldots \]
\[ e_k \rightarrow \{ \text{action}_k \} \]

Output: a program,

\[ \ldots \text{reading a maximal prefix } w \text{ from the input, that satisfies } e_1 | \ldots | e_k; \]
\[ \ldots \text{determining the minimal } i \text{, such that } w \in [e_i]; \]
\[ \ldots \text{executing action}_i \text{ for } w. \]
Implementation:

Idea:

- Create the DFA $P(A_e) = (Q, \Sigma, \delta, q_0, F)$ for the expression $e = (e_1 | \ldots | e_k)$;
- Define the sets:
  - $F_1 = \{ q \in F \mid q \cap \text{last}[e_1] \neq \emptyset \}$
  - $F_2 = \{ q \in F \setminus F_1 \mid q \cap \text{last}[e_2] \neq \emptyset \}$
  - $\ldots$
  - $F_k = \{ q \in (F \setminus \bigcup_{i=1}^{k} F_i) \mid q \cap \text{last}[e_k] \neq \emptyset \}$
- For input $w$ we find: $\delta^*(q_0, w) \in F_k$ if the scanner must execute action for $w$.

Implementation:

Idea (cont’d):

- The scanner manages two pointers $\langle A, B \rangle$ and the related states $\langle q_A, q_B \rangle$;
- Pointer $A$ points to the last position in the input, after which a state $q_A \in F$ was reached;
- Pointer $B$ tracks the current position.

Implementation:

Idea (cont’d):

- The current state being $q_B = \emptyset$, we consume input up to position $A$ and reset:
  - $B := A$; $A := \bot$;
  - $q_B := q_0$; $q_A := \bot$
Implementation:

Idea (cont'd):
- The current state being $q_B = \emptyset$, we consume input up to position $A$ and reset:

$$
B := A; \quad A := \bot;
$$

$$
q_B := q_0; \quad q_A := \bot
$$

-written ("Hello");

Input (generalized): a set of rules:

$$
\begin{align*}
\{ \text{state} \} & \quad \{ \text{action}_1 \text{ yybegin(state$_1$);} \} \\
\{ \text{action}_2 \text{ yybegin(state$_2$);} \} \\
\vdots \\
\{ \text{action}_k \text{ yybegin(state$_k$);} \} \\
\end{align*}
$$

- The statement $\text{yybegin}$ (state$_i$); resets the current state to state$_i$.
- The start state is called (e.g. flex JFlex) YYINITIAL.

... for example:

$$
\begin{align*}
\{ \text{YYINITIAL} \} & \quad \{ /* */ \text{ yybegin(COMMENT);} \} \\
\{ \text{yybegin(YYINITIAL)}; \} \\
\{ \text{} \text{ \textbackslash n} \} \\
\end{align*}
$$

Remarks:

- "." matches all characters different from "\n".
- For every state we generate the scanner respectively.
- Method $\text{yybegin}$ (STATE); switches between different scanners.
- Comments might be directly implemented as (admittedly overly complex) token-class.
- Scanner-states are especially handy for implementing preprocessors, expanding special fragments in regular programs.
Topic:

Syntactic Analysis
Syntactic Analysis

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

Discussion:

In general, parsers are not developed by hand, but generated from a specification:

- Such units may possibly be:
  - Expressions;
  - Statements;
  - Conditional branches;
  - loops; ...

Specification of the hierarchical structure: context-free grammars
Generated implementation: Pushdown automata + X
Chapter 1:
Basics of Context-free Grammars

Basics: Context-free Grammars

- Programs of programming languages can have arbitrary numbers of tokens, but only finitely many Token-classes.
- This is why we choose the set of Token-classes to be the finite alphabet of terminals $T$.
- The nested structure of program components can be described elegantly via context-free grammars...

Conventions

The rules of context-free grammars take the following form:

\[ A \rightarrow \alpha \quad \text{with} \quad A \in N, \quad \alpha \in (N \cup T)^* \]
Conventions

The rules of context-free grammars take the following form:

\[ A \rightarrow \alpha \quad \text{with} \quad A \in N, \quad \alpha \in (N \cup T)^* \]

... for example:

\[
\begin{align*}
S & \rightarrow aSb \\
S & \rightarrow \epsilon
\end{align*}
\]

Specified language: \( \{a^n b^n \mid n \geq 0\} \)

... further examples:

\[
\begin{align*}
S & \rightarrow \langle \text{stmt} \rangle \\
\langle \text{stmt} \rangle & \rightarrow \langle \text{if} \rangle \mid \langle \text{while} \rangle \mid \langle \text{rexp} \rangle \\
\langle \text{if} \rangle & \rightarrow \text{if} \ (\langle \text{rexp} \rangle) \ \langle \text{stmt} \rangle \ \text{else} \ \langle \text{stmt} \rangle \\
\langle \text{while} \rangle & \rightarrow \text{while} \ (\langle \text{rexp} \rangle) \ \langle \text{stmt} \rangle \\
\langle \text{rexp} \rangle & \rightarrow \text{int} \mid \langle \text{exp} \rangle \mid \langle \text{exp} \rangle = \langle \text{exp} \rangle \mid \ldots \\
\langle \text{exp} \rangle & \rightarrow \text{name} \mid \ldots
\end{align*}
\]

Conventions:

In examples, we specify nonterminals and terminals in general implicitly:

- nonterminals are: \( A, B, C, \ldots, \langle \text{exp} \rangle, \langle \text{stmt} \rangle, \ldots \);
- terminals are: \( a, b, c, \ldots, \text{int}, \text{name}, \ldots \);