

Compiler Construction

Lecture 3: Lexical Analysis II (Extended Matching Problem)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University

noll@cs.rwth-aachen.de

<http://www-i2.informatik.rwth-aachen.de/i2/cc12/>

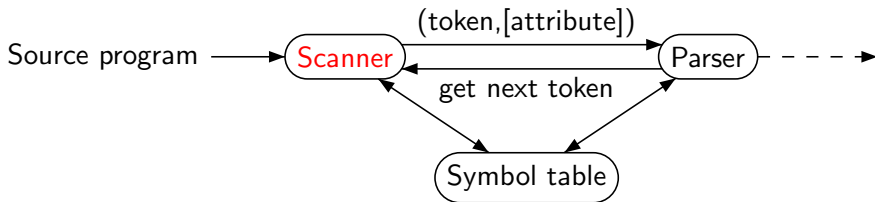
Summer Semester 2012

- 1 Repetition: Lexical Analysis
- 2 Complexity Analysis of Simple Matching
- 3 The Extended Matching Problem
- 4 First-Longest-Match Analysis
- 5 Implementation of FLM Analysis

Definition

The goal of **lexical analysis** is to decompose a source program into a sequence of lexemes and their transformation into a sequence of symbols.

The corresponding program is called a **scanner** (or **lexer**):



Example:

$\dots _x1_ := y2 + _1_ ; _ \dots$
 $\Downarrow$
 $\dots (id, p_1)(gets,)(id, p_2)(plus,)(int, 1)(sem,) \dots$

The DFA Method I

Known from *Formal Systems, Automata and Processes*:

Algorithm (DFA method)

Input: regular expression $\alpha \in RE_\Omega$, input string $w \in \Omega^*$

- Procedure:
- 1 using *Kleene's Theorem*, construct $\mathfrak{A}_\alpha \in NFA_\Omega$ such that $L(\mathfrak{A}_\alpha) = \llbracket \alpha \rrbracket$
 - 2 apply *powerset construction* to obtain $\mathfrak{A}'_\alpha = \langle Q', \Omega, \delta', q'_0, F' \rangle \in DFA_\Omega$ with $L(\mathfrak{A}'_\alpha) = L(\mathfrak{A}_\alpha) = \llbracket \alpha \rrbracket$
 - 3 solve the *matching problem* by deciding whether $\hat{\delta}'(q'_0, w) \in F'$

Output: “yes” or “no”

Example

$\alpha := a^*b \mid a^*$ (cf. Example 2.8)

The DFA Method II

The powerset construction involves the following concept:

Definition (ε -closure)

Let $\mathfrak{A} = \langle Q, \Omega, \delta, q_0, F \rangle \in NFA_{\Omega}$. The ε -closure $\varepsilon(T) \subseteq Q$ of a subset $T \subseteq Q$ is defined by

- $T \subseteq \varepsilon(T)$ and
- if $q \in \varepsilon(T)$, then $\delta(q, \varepsilon) \subseteq \varepsilon(T)$

Definition (Powerset construction)

Let $\mathfrak{A} = \langle Q, \Omega, \delta, q_0, F \rangle \in NFA_{\Omega}$. The **powerset automaton** $\mathfrak{A}' = \langle Q', \Omega, \delta', q'_0, F' \rangle \in DFA_{\Omega}$ is defined by

- $Q' := 2^Q$
- $\forall T \subseteq Q, a \in \Omega : \delta'(T, a) := \varepsilon \left(\bigcup_{q \in T} \delta(q, a) \right)$
- $q'_0 := \varepsilon(\{q_0\})$
- $F' := \{T \subseteq Q \mid T \cap F \neq \emptyset\}$

- 1 Repetition: Lexical Analysis
- 2 Complexity Analysis of Simple Matching
- 3 The Extended Matching Problem
- 4 First-Longest-Match Analysis
- 5 Implementation of FLM Analysis

① in construction phase:

- **Kleene method:** time and space $\mathcal{O}(|\alpha|)$
(where $|\alpha| := \text{length of } \alpha$)
- **Powerset construction:** time and space $\mathcal{O}(2^{|\mathfrak{A}_\alpha|}) = \mathcal{O}(2^{|\alpha|})$
(where $|\mathfrak{A}_\alpha| := \# \text{ of states of } \mathfrak{A}_\alpha$)

② at runtime:

- **Word problem:** time $\mathcal{O}(|w|)$ (where $|w| := \text{length of } w$),
space $\mathcal{O}(1)$ (but $\mathcal{O}(2^{|\alpha|})$ for storing DFA)

⇒ nice runtime behavior but memory requirements very high
(and exponential time in construction phase)

The NFA Method

Idea: reduce memory requirements by **applying powerset construction at runtime**, i.e., only “to the run of w through $\mathfrak{A}_\alpha$ ”

Algorithm 3.1 (NFA method)

Input: *automaton* $\mathfrak{A}_\alpha = \langle Q, \Omega, \delta, q_0, F \rangle \in \text{NFA}_\Omega$,
input string $w \in \Omega^*$

Variables: $T \subseteq Q$, $a \in \Omega$

Procedure: $T := \varepsilon(\{q_0\})$;
 while $w \neq \varepsilon$ **do**
 $a := \text{head}(w)$;
 $T := \varepsilon\left(\bigcup_{q \in T} \delta(q, a)\right)$;
 $w := \text{tail}(w)$
 od

Output: *if* $T \cap F \neq \emptyset$ *then* “yes” *else* “no”

For NFA method at runtime:

- Space: $\mathcal{O}(|\alpha|)$ (for storing NFA and T)
- Time: $\mathcal{O}(|\alpha| \cdot |w|)$
(in the loop's body, $|T|$ states need to be considered)

⇒ trades exponential space for increase in time

Comparison:

Method	Space	Time (for " $w \in \llbracket \alpha \rrbracket ?$ ")
DFA	$\mathcal{O}(2^{ \alpha })$	$\mathcal{O}(w)$
NFA	$\mathcal{O}(\alpha)$	$\mathcal{O}(\alpha \cdot w)$

In practice:

- Exponential blowup of DFA method usually does not occur in “real” applications (⇒ used in `[f]lex`)
- Improvement of NFA method: caching of transitions $\delta'(T, a)$
⇒ combination of both methods

- 1 Repetition: Lexical Analysis
- 2 Complexity Analysis of Simple Matching
- 3 The Extended Matching Problem**
- 4 First-Longest-Match Analysis
- 5 Implementation of FLM Analysis

The Extended Matching Problem I

Definition 3.2

Let $n \geq 1$ and $\alpha_1, \dots, \alpha_n \in RE_\Omega$ with $\varepsilon \notin \llbracket \alpha_i \rrbracket \neq \emptyset$ for every $i \in [n]$ ($= \{1, \dots, n\}$). Let $\Sigma := \{T_1, \dots, T_n\}$ be an alphabet of corresponding **tokens** and $w \in \Omega^+$. If $w_1, \dots, w_k \in \Omega^+$ such that

- $w = w_1 \dots w_k$ and
- for every $j \in [k]$ there exists $i_j \in [n]$ such that $w_j \in \llbracket \alpha_{i_j} \rrbracket$,

then

- $(w_1, \dots, w_k)$ is called a **decomposition** and
- $(T_{i_1}, \dots, T_{i_k})$ is called an **analysis**

of w w.r.t. $\alpha_1, \dots, \alpha_n$.

Problem 3.3 (Extended matching problem)

Given $\alpha_1, \dots, \alpha_n \in RE_\Omega$ and $w \in \Omega^+$, decide whether there exists a decomposition of w w.r.t. $\alpha_1, \dots, \alpha_n$ and determine a corresponding analysis.

Observation: neither the decomposition nor the analysis are uniquely determined

Example 3.4

- ① $\alpha = a^+, w = aa$
 $\Rightarrow$ two decompositions (aa) and (a, a) with respective (unique) analyses (T_1) and (T_1, T_1)
- ② $\alpha_1 = a \mid b, \alpha_2 = a \mid c, w = a$
 $\Rightarrow$ unique decomposition (a) but two analyses (T_1) and (T_2)

Goal: make both unique $\Rightarrow$ deterministic scanning

- 1 Repetition: Lexical Analysis
- 2 Complexity Analysis of Simple Matching
- 3 The Extended Matching Problem
- 4 First-Longest-Match Analysis**
- 5 Implementation of FLM Analysis

Two principles:

- ❶ **Principle of the longest match** (“maximal munch tokenization”)
 - for uniqueness of decomposition
 - make lexemes as long as possible
 - motivated by applications: e.g., every (non-empty) prefix of an identifier is also an identifier
- ❷ **Principle of the first match**
 - for uniqueness of analysis
 - choose first matching regular expression (in the given order)
 - therefore: arrange keywords before identifiers (if keywords protected)

Principle of the Longest Match

Definition 3.5 (Longest-match decomposition)

A decomposition $(w_1, \dots, w_k)$ of $w \in \Omega^+$ w.r.t. $\alpha_1, \dots, \alpha_n \in RE_\Omega$ is called a **longest-match decomposition (LM decomposition)** if, for every $i \in [k]$, $x \in \Omega^+$, and $y \in \Omega^*$,

$$w = w_1 \dots w_i x y \implies \text{there is no } j \in [n] \text{ such that } w_i x \in \llbracket \alpha_j \rrbracket$$

Corollary 3.6

Given w and $\alpha_1, \dots, \alpha_n$,

- *at most one LM decomposition of w exists (clear by definition) and*
- *it is possible that w has a decomposition but no LM decomposition (see following example).*

Example 3.7

$w = aab$, $\alpha_1 = a^+$, $\alpha_2 = ab$

$\implies (a, ab)$ is a decomposition but no LM decomposition exists

Principle of the First Match

Problem: a (unique) LM decomposition can have **several associated analyses** (since $\llbracket \alpha_i \rrbracket \cap \llbracket \alpha_j \rrbracket \neq \emptyset$ with $i \neq j$ is possible; cf. keyword/identifier problem)

Definition 3.8 (First-longest-match analysis)

Let $(w_1, \dots, w_k)$ be the LM decomposition of $w \in \Omega^+$ w.r.t. $\alpha_1, \dots, \alpha_n \in RE_\Omega$. Its **first-longest-match analysis (FLM analysis)** $(T_{i_1}, \dots, T_{i_k})$ is determined by

$$i_j := \min\{l \in [n] \mid w_j \in \llbracket \alpha_l \rrbracket\}$$

for every $j \in [k]$.

Corollary 3.9

Given w and $\alpha_1, \dots, \alpha_n$, there is at most one FLM analysis of w . It exists iff the LM decomposition of w exists.

- 1 Repetition: Lexical Analysis
- 2 Complexity Analysis of Simple Matching
- 3 The Extended Matching Problem
- 4 First-Longest-Match Analysis
- 5 Implementation of FLM Analysis

Algorithm 3.10 (FLM analysis—overview)

Input: expressions $\alpha_1, \dots, \alpha_n \in RE_\Omega$, tokens $\{T_1, \dots, T_n\}$,
input word $w \in \Omega^+$

Procedure:

- 1 for every $i \in [n]$, construct $\mathfrak{A}_i \in DFA_\Omega$ such that $L(\mathfrak{A}_i) = \llbracket \alpha_i \rrbracket$ (see *DFA method*; Alg. 2.9)
- 2 construct the *product automaton* $\mathfrak{A} \in DFA_\Omega$ such that $L(\mathfrak{A}) = \bigcup_{i=1}^n \llbracket \alpha_i \rrbracket$
- 3 *partition the set of final states* of $\mathfrak{A}$ to follow the *first-match principle*
- 4 extend the resulting DFA to a *backtracking DFA* which implements the *longest-match principle*, and let it run on w

Output: FLM analysis of w (if existing)

(2) The Product Automaton

Definition 3.11 (Product automaton)

Let $\mathfrak{A}_i = \langle Q_i, \Omega, \delta_i, q_0^{(i)}, F_i \rangle \in \text{DFA}_\Omega$ for every $i \in [n]$. The **product automaton** $\mathfrak{A} = \langle Q, \Omega, \delta, q_0, F \rangle \in \text{DFA}_\Omega$ is defined by

- $Q := Q_1 \times \dots \times Q_n$
- $q_0 := (q_0^{(1)}, \dots, q_0^{(n)})$
- $\delta((q^{(1)}, \dots, q^{(n)}), a) := (\delta_1(q^{(1)}, a), \dots, \delta_n(q^{(n)}, a))$
- $(q^{(1)}, \dots, q^{(n)}) \in F$ iff there ex. $i \in [n]$ such that $q^{(i)} \in F_i$

Lemma 3.12

The above construction yields $L(\mathfrak{A}) = \bigcup_{i=1}^n L(\mathfrak{A}_i)$ ($= \bigcup_{i=1}^n \llbracket \alpha_i \rrbracket$).

Remark: similar construction for intersection ($F := F_1 \times \dots \times F_n$)

(3) Partitioning the Final States

Definition 3.13 (Partitioning of final states)

Let $\mathfrak{A} = \langle Q, \Omega, \delta, q_0, F \rangle \in DFA_\Omega$ be the product automaton as constructed before. Its set of final states is **partitioned** into $F = \bigsqcup_{i=1}^n F^{(i)}$ by the requirement

$$(q^{(1)}, \dots, q^{(n)}) \in F^{(i)} \iff q^{(i)} \in F_i \text{ and } \forall j \in [i-1] : q^{(j)} \notin F_j$$

(or: $F^{(i)} := (Q_1 \setminus F_1) \times \dots \times (Q_{i-1} \setminus F_{i-1}) \times F_i \times Q_{i+1} \times \dots \times Q_n$)

Corollary 3.14

The above construction yields ($w \in \Omega^+$, $i \in [n]$):

$$\hat{\delta}(q_0, w) \in F^{(i)} \text{ iff } w \in \llbracket \alpha_i \rrbracket \text{ and } w \notin \bigcup_{j=1}^{i-1} \llbracket \alpha_j \rrbracket.$$

Definition 3.15 (Productive states)

Given $\mathfrak{A}$ as above, $q \in Q$ is called **productive** if there exists $w \in \Omega^*$ such that $\hat{\delta}(q, w) \in F$. Notation: productive states $P \subseteq Q$ (thus $F \subseteq P$).

(4) The Backtracking DFA I

Goal: extend $\mathfrak{A}$ to the backtracking DFA $\mathfrak{B}$ with output by equipping the input tape with two pointers: a **backtracking head** for marking the last encountered match, and a **lookahead** for determining the longest match.

A configuration of $\mathfrak{B}$ has three components
(remember: $\Sigma := \{T_1, \dots, T_n\}$ denotes the set of tokens):

- ① a **mode** $m \in \{N\} \uplus \Sigma$:
 - $m = N$ (“normal”): look for first match (no final state reached yet)
 - $m = T \in \Sigma$: token T has been recognized, looking for possible longer match
- ② an **input tape** $vqw \in \Omega^* \cdot Q \cdot \Omega^*$:
 - v : lookahead part of input ($v \neq \varepsilon \implies m \in \Sigma$)
 - q : current state of $\mathfrak{A}$
 - w : remaining input
- ③ an **output tape** $W \in \Sigma^* \cdot \{\varepsilon, \text{lexerr}\}$:
 - Σ^* : sequence of tokens recognized so far
 - **lexerr**: a lexical error has occurred (i.e., a non-productive state was entered or the suffix of the input is not a valid lexeme)

(4) The Backtracking DFA II

Definition 3.16 (Backtracking DFA)

- The set of **configurations** of $\mathfrak{B}$ is given by

$$(\{N\} \uplus \Sigma) \times \Omega^* \cdot Q \cdot \Omega^* \times \Sigma^* \cdot \{\varepsilon, \text{lexerr}\}$$

- The **initial configuration** for an input word $w \in \Omega^+$ is (N, q_0w, ε) .
- The **transitions** of $\mathfrak{B}$ are defined as follows (where $q' := \delta(q, a)$):
 - normal mode: look for a match

$$(N, qaw, W) \vdash \begin{cases} (T_i, q'w, W) & \text{if } q' \in F^{(i)} \\ (N, q'w, W) & \text{if } q' \in P \setminus F \\ \textbf{output: } W \cdot \text{lexerr} & \text{if } q' \notin P \end{cases}$$

- backtrack mode: look for longest match

$$(T, vqaw, W) \vdash \begin{cases} (T_i, q'w, W) & \text{if } q' \in F^{(i)} \\ (T, vaq'w, W) & \text{if } q' \in P \setminus F \\ (N, q_0vaw, WT) & \text{if } q' \notin P \end{cases}$$

- end of input

$$\begin{aligned} (T, q, W) &\vdash \textbf{output: } WT && \text{if } q \in F \\ (N, q, W) &\vdash \textbf{output: } W \cdot \text{lexerr} && \text{if } q \in P \setminus F \\ (T, vaq, W) &\vdash (N, q_0va, WT) && \text{if } q \in P \setminus F \end{aligned}$$

(4) The Backtracking DFA III

Lemma 3.17

Given the backtracking DFA $\mathfrak{B}$ as before and $w \in \Omega^+$,

$$(N, q_0 w, \varepsilon) \vdash^* \begin{cases} W \in \Sigma^* & \text{iff } W \text{ is the FLM analysis of } w \\ W \cdot \text{lexerr} & \text{iff no FLM analysis of } w \text{ exists} \end{cases}$$

Example 3.18

$\alpha = (ab)^+$, $w = abaa$ (on the board)

(4) The Backtracking DFA IV

Remarks:

- **Time complexity:** $\mathcal{O}(|w|^2)$ in worst case

Example 3.19

$\alpha_1 = a$, $\alpha_2 = a^*b$, $w = a^m$ requires $\mathcal{O}(m^2)$

- Improvement by **tabular method** (similar to Knuth-Morris-Pratt Algorithm for pattern matching in strings)

Literature: Th. Reps: *“Maximal-Munch” Tokenization in Linear Time*, ACM TOPLAS 20(2), 1998, 259–273