

Compiler Construction

Lecture 3: Lexical Analysis II

(The 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/cc10/>

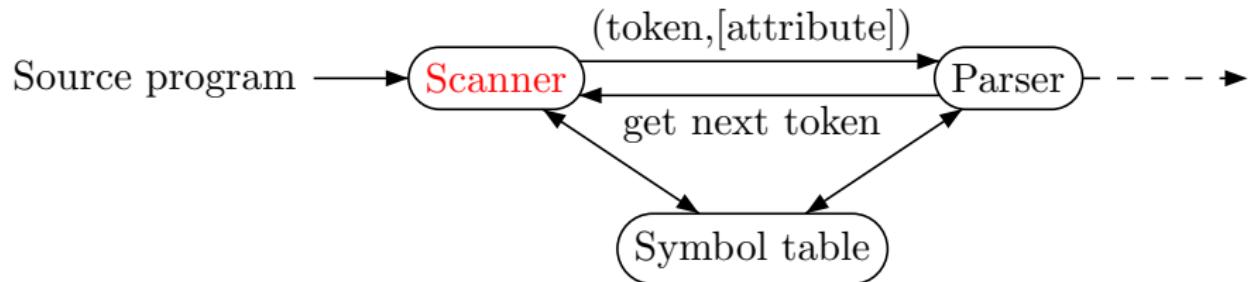
Winter semester 2010/11

- 1 Repetition: Lexical Analysis
- 2 The Simple Matching Problem
- 3 The Extended Matching Problem

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**:



Example:

... $\underline{x1} := y2 + \underline{1}; \underline{\dots}$
 \Downarrow
... (id, p₁)(gets,)(id, p₂)(plus,)(int, 1)(sem,) ...

- 1 Repetition: Lexical Analysis
- 2 The Simple Matching Problem
- 3 The Extended Matching Problem

The Simple Matching Problem I

Problem 3.1 (Simple matching problem)

Given $\alpha \in RE_\Omega$ and $w \in \Omega^*$, decide whether $w \in \llbracket \alpha \rrbracket$ or not.

This problem can be solved using the following concept:

Definition 3.2 (Finite automaton)

A **nondeterministic finite automaton (NFA)** is of the form

$\mathfrak{A} = \langle Q, \Omega, \delta, q_0, F \rangle$ where

- Q is a finite set of **states**
- Ω denotes the **input alphabet**
- $\delta : Q \times \Omega_\varepsilon \rightarrow 2^Q$ is the **transition function** where $\Omega_\varepsilon := \Omega \cup \{\varepsilon\}$
(notation: $q \xrightarrow{x} q'$ for $q' \in \delta(q, x)$)
- $q_0 \in Q$ is the **initial state**
- $F \subseteq Q$ is the set of **final states**

The set of all NFA over Ω is denoted by NFA_Ω .

If $\delta(q, \varepsilon) = \emptyset$ and $|\delta(q, a)| = 1$ for every $q \in Q$ and $a \in \Omega$ (i.e., $\delta : Q \times \Omega \rightarrow Q$), then \mathfrak{A} is called **deterministic (DFA)**. Notation: DFA_Ω

The Simple Matching Problem II

Definition 3.3 (Acceptance condition)

Let $\mathfrak{A} = \langle Q, \Omega, \delta, q_0, F \rangle \in NFA_{\Omega}$ and $w = a_1 \dots a_n \in \Omega^*$.

- A w -labeled **\mathfrak{A} -run** from q_1 to q_2 is a sequence of transitions

$$q_1 \xrightarrow{\varepsilon^*} \xrightarrow{a_1} \xrightarrow{\varepsilon^*} \xrightarrow{a_2} \xrightarrow{\varepsilon^*} \dots \xrightarrow{\varepsilon^*} \xrightarrow{a_n} \xrightarrow{\varepsilon^*} q_2$$

- \mathfrak{A} **accepts** w if there is a w -labeled \mathfrak{A} -run from q_0 to some $q \in F$
- The **language recognized** by \mathfrak{A} is

$$L(\mathfrak{A}) := \{w \in \Omega^* \mid \mathfrak{A} \text{ accepts } w\}$$

- A language $L \subseteq \Omega^*$ is called **NFA-recognizable** if there exists a NFA \mathfrak{A} such that $L(\mathfrak{A}) = L$

Example 3.4

NFA for $a^*b \mid a^*$ (on the board)

Remarks:

- NFA as specified in Definition 3.2 are sometimes called **NFA with ε -transitions (ε -NFA)**.
- For $\mathfrak{A} \in DFA_{\Omega}$, the acceptance condition yields $\widehat{\delta} : Q \times \Omega^* \rightarrow Q$ with $\widehat{\delta}(q, \varepsilon) = q$ and $\widehat{\delta}(q, aw) = \widehat{\delta}(\delta(q, a), w)$, and

$$L(\mathfrak{A}) = \{w \in \Omega^* \mid \widehat{\delta}(q_0, w) \in F\}.$$

Known from *Automata Theory and Formal Languages*:

Algorithm 3.5 (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 $\widehat{\delta}'(q'_0, w) \in F'$

Output: “yes” or “no”

The powerset construction involves the following concept:

Definition 3.6 (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 $\delta(q, \varepsilon) \subseteq \varepsilon(T)$

Example 3.7

- ➊ Kleene's Theorem (in general)
- ➋ Powerset construction
(for NFA \mathfrak{A}_α with $\alpha := a^*b \mid a^*$; cf. Example 3.4)

(on the board)

① In construction phase:

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

② At runtime:

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

⇒ Nice runtime behavior but memory requirements too high
(and exponential time in construction phase)

The NFA Method

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

Algorithm 3.8 (NFA method)

Input: automaton $\mathfrak{A}_\alpha = \langle Q, \Omega, \delta, q_0, F \rangle \in NFA_\Omega$,
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”

Example 3.9 (cf. Example 3.4)

NFA \mathfrak{A}_α for $\alpha := a^*b \mid a^*$, $w := aab$

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. Here evaluation of $\delta(q, a)$ is considered as *one* operation.)

⇒ 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 `f1ex`)
- Improvement of NFA method: caching of transitions $\widehat{\delta}(T, a)$
⇒ combination of both methods

- 1 Repetition: Lexical Analysis
- 2 The Simple Matching Problem
- 3 The Extended Matching Problem

The Extended Matching Problem I

Definition 3.10

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 3.11 (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 3.12

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

Goal: make both unique \implies **deterministic scanning**