



Compiler Construction

Lecture 3: Lexical Analysis II (Extended Matching Problem)

Winter Semester 2018/19

Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

<https://moves.rwth-aachen.de/teaching/ws-1819/cc/>

Recap: Lexical Analysis

Outline of Lecture 3

Recap: Lexical Analysis

The Extended Matching Problem

First-Longest-Match Analysis

Implementation of FLM Analysis

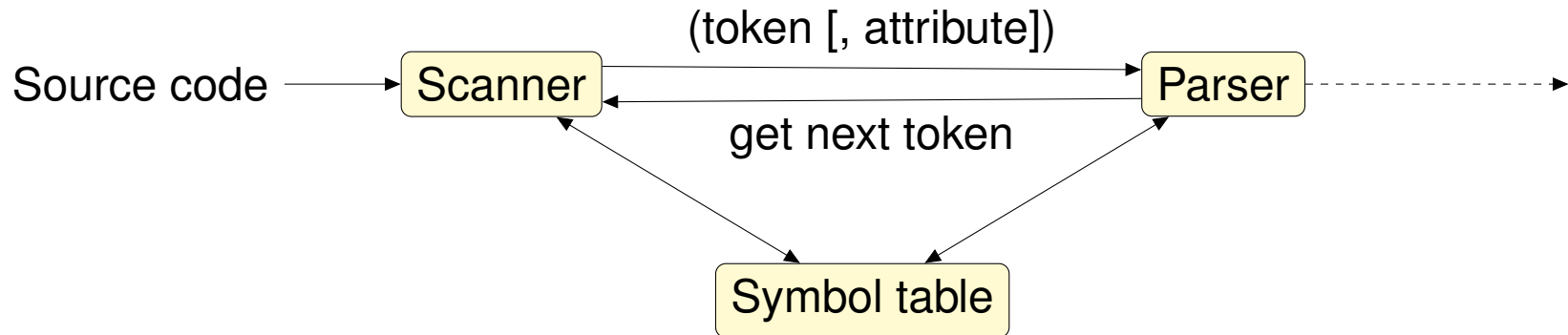
Recap: Lexical Analysis

Lexical Analysis

Definition

The goal of **lexical analysis** is the decomposition 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:

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

Recap: Lexical Analysis

Regular Expressions

Definition (Syntax of regular expressions)

Given some alphabet Ω , the set of **regular expressions** over Ω , RE_Ω , is the least set with

- $\emptyset \in RE_\Omega$,
- $\Omega \subseteq RE_\Omega$, and
- whenever $\alpha, \beta \in RE_\Omega$, also $\alpha \mid \beta, \alpha \cdot \beta, \alpha^* \in RE_\Omega$.

Remarks:

- abbreviations: $\alpha^+ := \alpha \cdot \alpha^*$, $\varepsilon := \emptyset^*$
- $\alpha \cdot \beta$ often written as $\alpha\beta$
- Binding priority: $* > \cdot > \mid$ (i.e., $a \mid b \cdot c^* := a \mid (b \cdot (c^*))$)

Recap: Lexical Analysis

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** (cf. Definition 2.11) 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 $\delta'^*(q'_0, w) \in F'$

Output: “yes” or “no”

Recap: Lexical Analysis

The DFA Method II

The powerset construction involves the following concept:

Definition (ε -closure)

Let $\mathcal{A} = \langle Q, \Omega, \delta, q_0, F \rangle \in NFA_{\Omega}$. The ε -closure $\varepsilon(T) \subseteq Q$ of a subset $T \subseteq Q$ is the least set with (1) $T \subseteq \varepsilon(T)$ and (2) if $q \in \varepsilon(T)$, then $\delta(q, \varepsilon) \subseteq \varepsilon(T)$

Definition (Powerset construction)

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

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

The Extended Matching Problem

Outline of Lecture 3

Recap: Lexical Analysis

The Extended Matching Problem

First-Longest-Match Analysis

Implementation of FLM Analysis

The Extended Matching Problem

The Extended Matching Problem I

Definition 3.1

Let $n \geq 1$ and $\alpha_1, \dots, \alpha_n \in RE_\Omega$ with $\varepsilon \notin [[\alpha_i]] \neq \emptyset$ for every $i \in [n]$ (where $[n] := \{1, \dots, n\}$). 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 $w_j \in [[\alpha_{i_j}]]$,

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. $\alpha_1, \dots, \alpha_n$.

The Extended Matching Problem

The Extended Matching Problem I

Definition 3.1

Let $n \geq 1$ and $\alpha_1, \dots, \alpha_n \in RE_\Omega$ with $\varepsilon \notin [[\alpha_i]] \neq \emptyset$ for every $i \in [n]$ (where $[n] := \{1, \dots, n\}$). 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 $w_j \in [[\alpha_{i_j}]]$,

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 3.2 (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.

The Extended Matching Problem

The Extended Matching Problem II

Observation: neither the decomposition nor the analysis are uniquely determined

Example 3.3

1. – $n = 1, \alpha_1 = a^+$

– $w = aa$

⇒ two decompositions (aa) and (a, a) with respective (unique) analyses (T_1) and (T_1, T_1)

The Extended Matching Problem

The Extended Matching Problem II

Observation: neither the decomposition nor the analysis are uniquely determined

Example 3.3

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

The Extended Matching Problem

The Extended Matching Problem II

Observation: neither the decomposition nor the analysis are uniquely determined

Example 3.3

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

Goal: make both unique \implies deterministic scanning

First-Longest-Match Analysis

Outline of Lecture 3

Recap: Lexical Analysis

The Extended Matching Problem

First-Longest-Match Analysis

Implementation of FLM Analysis

Ensuring Uniqueness

Two principles

1. **Principle of the longest match** (“maximal munch tokenisation”)
 - for uniqueness of decomposition
 - make lexemes as long as possible
 - motivated by practical considerations:
 - every proper prefix of an identifier is also an identifier
 - real-valued constants contain integer constants as prefixes

First-Longest-Match Analysis

Ensuring Uniqueness

Two principles

1. **Principle of the longest match** (“maximal munch tokenisation”)
 - for uniqueness of decomposition
 - make lexemes as long as possible
 - motivated by practical considerations:
 - every proper prefix of an identifier is also an identifier
 - real-valued constants contain integer constants as prefixes
2. **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)

First-Longest-Match Analysis

Ensuring Uniqueness

Two principles

1. Principle of the longest match (“maximal munch tokenisation”)

- for uniqueness of decomposition
- make lexemes as long as possible
- motivated by practical considerations:
 - every proper prefix of an identifier is also an identifier
 - real-valued constants contain integer constants as prefixes

2. 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)

Remark: uniqueness of analysis could also be achieved by requiring **disjointness** of symbol classes (i.e., $[[\alpha_i]] \cap [[\alpha_j]] = \emptyset$ for $i \neq j$)

- for example, $Id := (a | \dots)(a | \dots | 0 | \dots)^* \setminus \{if, while, begin, \dots\}$
- **but:** expensive implementation due to product construction ($A \setminus B = A \cap \bar{B}$; cf. Def. 3.10)

First-Longest-Match Analysis

Principle of the Longest Match

Definition 3.4 (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 (LM) decomposition** if, for every $i \in [k]$, $x \in \Omega^+$, and $y \in \Omega^*$,

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

First-Longest-Match Analysis

Principle of the Longest Match

Definition 3.4 (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 (LM) decomposition** if, for every $i \in [k]$, $x \in \Omega^+$, and $y \in \Omega^*$,

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

Corollary 3.5

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

- *at most one LM decomposition of w exists (clear by definition) and*

First-Longest-Match Analysis

Principle of the Longest Match

Definition 3.4 (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 (LM) decomposition** if, for every $i \in [k]$, $x \in \Omega^+$, and $y \in \Omega^*$,

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

Corollary 3.5

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 3.6

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

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

First-Longest-Match Analysis

Principle of the First Match

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

First-Longest-Match Analysis

Principle of the First Match

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

Definition 3.7 (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

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

for every $j \in [k]$.

First-Longest-Match Analysis

Principle of the First Match

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

Definition 3.7 (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

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

for every $j \in [k]$.

Corollary 3.8

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.

Implementation of FLM Analysis

Outline of Lecture 3

Recap: Lexical Analysis

The Extended Matching Problem

First-Longest-Match Analysis

Implementation of FLM Analysis

Implementation of FLM Analysis

Implementation of FLM Analysis

Algorithm 3.9 (FLM analysis – overview)

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

Implementation of FLM Analysis

Implementation of FLM Analysis

Algorithm 3.9 (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 $\mathcal{A}_i \in DFA_\Omega$ such that $L(\mathcal{A}_i) = \llbracket \alpha_i \rrbracket$
(see *DFA method*; Algorithm 2.9)

Implementation of FLM Analysis

Implementation of FLM Analysis

Algorithm 3.9 (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*; Algorithm 2.9)

2. construct the *product automaton* $\mathfrak{A} \in DFA_\Omega$ such that $L(\mathfrak{A}) = \bigcup_{i=1}^n \llbracket \alpha_i \rrbracket$

Implementation of FLM Analysis

Implementation of FLM Analysis

Algorithm 3.9 (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**; Algorithm 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***

Implementation of FLM Analysis

Implementation of FLM Analysis

Algorithm 3.9 (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*; Algorithm 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*

Implementation of FLM Analysis

Implementation of FLM Analysis

Algorithm 3.9 (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*; Algorithm 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

5. let the *backtracking DFA* run on w

Implementation of FLM Analysis

Implementation of FLM Analysis

Algorithm 3.9 (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*; Algorithm 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

5. let the *backtracking DFA* run on w

Output: FLM analysis of w (if existing)

(2) The Product Automaton

Definition 3.10 (Product automaton)

Let $\mathcal{A}_i = \langle Q_i, \Omega, \delta_i, q_0^{(i)}, F_i \rangle \in DFA_\Omega$ for every $i \in [n]$. The **product automaton** $\mathcal{A} = \langle Q, \Omega, \delta, q_0, F \rangle \in DFA_\Omega$ 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 $q^{(i)} \in F_i$

(2) The Product Automaton

Definition 3.10 (Product automaton)

Let $\mathfrak{A}_i = \langle Q_i, \Omega, \delta_i, q_0^{(i)}, F_i \rangle \in DFA_\Omega$ for every $i \in [n]$. The **product automaton** $\mathfrak{A} = \langle Q, \Omega, \delta, q_0, F \rangle \in DFA_\Omega$ 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 3.11

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

(2) The Product Automaton

Definition 3.10 (Product automaton)

Let $\mathfrak{A}_i = \langle Q_i, \Omega, \delta_i, q_0^{(i)}, F_i \rangle \in DFA_\Omega$ for every $i \in [n]$. The **product automaton** $\mathfrak{A} = \langle Q, \Omega, \delta, q_0, F \rangle \in DFA_\Omega$ 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 3.11

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 ($F := F_1 \times \dots \times F_n$)

Implementation of FLM Analysis

(3) Partitioning the Final States

Definition 3.12 (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

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

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

Implementation of FLM Analysis

(3) Partitioning the Final States

Definition 3.12 (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

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

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

Corollary 3.13

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

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

Implementation of FLM Analysis

(3) Partitioning the Final States

Definition 3.12 (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

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

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

Corollary 3.13

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

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

Definition 3.14 (Productive states)

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

(4) The Backtracking DFA I

Goal: extend \mathcal{A} to the backtracking DFA \mathcal{B} with output by equipping the input tape with two pointers: a **backtracking head** for marking the last encountered match, and a **lookahead** for (possibly) extending it to the longest match.

(4) The Backtracking DFA I

Goal: extend \mathcal{A} to the backtracking DFA \mathcal{B} with output by equipping the input tape with two pointers: a **backtracking head** for marking the last encountered match, and a **lookahead** for (possibly) extending it to the longest match.

A **configuration** of \mathcal{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 initial match (no final state reached yet)
 - $m = T \in \Sigma$: token T has been recognised, looking for possible longer match

(4) The Backtracking DFA I

Goal: extend \mathcal{A} to the backtracking DFA \mathcal{B} with output by equipping the input tape with two pointers: a **backtracking head** for marking the last encountered match, and a **lookahead** for (possibly) extending it to the longest match.

A **configuration** of \mathcal{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 initial match (no final state reached yet)
 - $m = T \in \Sigma$: token T has been recognised, 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 \mathcal{A}
 - w : remaining input

Implementation of FLM Analysis

(4) The Backtracking DFA I

Goal: extend \mathcal{A} to the backtracking DFA \mathcal{B} with output by equipping the input tape with two pointers: a **backtracking head** for marking the last encountered match, and a **lookahead** for (possibly) extending it to the longest match.

A **configuration** of \mathcal{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 initial match (no final state reached yet)
 - $m = T \in \Sigma$: token T has been recognised, 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 \mathcal{A}
 - w : remaining input
3. an **output tape** $W \in \Sigma^* \cdot \{\varepsilon, \text{lexerr}\}$:
 - Σ^* : sequence of tokens recognised so far
 - **lexerr**: a lexical error has occurred (i.e., a non-productive state was entered or the suffix of the recognised part of the input is not a valid lexeme)

(4) The Backtracking DFA II

Definition 3.15 (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 $(N, q_0 w, \varepsilon)$.

(4) The Backtracking DFA II

Definition 3.15 (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 $(N, q_0 w, \varepsilon)$.
- The **transitions** of \mathfrak{B} are defined as follows (where $q' := \delta(q, a)$):
 - normal mode: look for initial match

$$(N, qaw, W) \vdash \begin{cases} (T_i, q'w, W) & \text{if } q' \in F^{(i)} & (1) \\ (N, q'w, W) & \text{if } q' \in P \setminus F & (2) \\ \mathbf{output: } W \cdot \text{lexerr} & \text{if } q' \notin P & (3) \end{cases}$$

(4) The Backtracking DFA II

Definition 3.15 (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 initial match

$$(N, qaw, W) \vdash \begin{cases} (T_i, q'w, W) & \text{if } q' \in F^{(i)} & (1) \\ (N, q'w, W) & \text{if } q' \in P \setminus F & (2) \\ \text{output: } W \cdot \text{lexerr} & \text{if } q' \notin P & (3) \end{cases}$$

– backtrack mode: look for longest match

$$(T, vqaw, W) \vdash \begin{cases} (T_i, q'w, W) & \text{if } q' \in F^{(i)} & (4) \\ (T, vaq'w, W) & \text{if } q' \in P \setminus F & (5) \\ (N, q_0vaw, WT) & \text{if } q' \notin P & (6) \end{cases}$$

Implementation of FLM Analysis

(4) The Backtracking DFA II

Definition 3.15 (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 initial match

$$(N, qaw, W) \vdash \begin{cases} (T_i, q'w, W) & \text{if } q' \in F^{(i)} & (1) \\ (N, q'w, W) & \text{if } q' \in P \setminus F & (2) \\ \text{output: } W \cdot \text{lexerr} & \text{if } q' \notin P & (3) \end{cases}$$

- backtrack mode: look for longest match

$$(T, vqaw, W) \vdash \begin{cases} (T_i, q'w, W) & \text{if } q' \in F^{(i)} & (4) \\ (T, vaq'w, W) & \text{if } q' \in P \setminus F & (5) \\ (N, q_0vaw, WT) & \text{if } q' \notin P & (6) \end{cases}$$

- end of input:

$$\begin{aligned} (T, q, W) &\vdash \text{output: } WT && \text{if } q \in F && (7) \\ (N, q, W) &\vdash \text{output: } W \cdot \text{lexerr} && \text{if } q \in P \setminus F && (8) \\ (T, vaq, W) &\vdash (N, q_0va, WT) && \text{if } q \in P \setminus F && (9) \end{aligned}$$

Implementation of FLM Analysis

(4) The Backtracking DFA III

Lemma 3.16

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

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

Proof.

(omitted) □

Implementation of FLM Analysis

(4) The Backtracking DFA III

Lemma 3.16

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

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

Proof.

(omitted) □

Example 3.17

- $\Omega = \{a, b\}$, $w = baa$
- $n = 3$, $\alpha_1 = a$, $\alpha_2 = a^+b$, $\alpha_3 = b$, $\Sigma = \{T_1, T_2, T_3\}$

(on the board)