Introduction to Formal Language Theory — day 2

Regular Languages

Wiebke Petersen & Kata Balogh

(Heinrich-Heine-Universität Düsseldorf)

NASSLLI 2014
University of Maryland, College Park
Outline

1. repetition
2. right-linear grammars
3. regular expressions
4. finite-state automata
5. Theorem of Kleene
Recall: basic definitions

- **alphabet** $\Sigma$: nonempty, finite set of **symbols**
- **word** $w$: a finite string $x_1 \ldots x_n$ of symbols; $(x_1 \ldots x_n \in \Sigma)$
- **length** of a word $|w|$: number of symbols of a word $w$ (example: $|abbaca| = 6$)
- **empty word** $\epsilon$: the word of length 0
- $\Sigma^*$ is the set of all words over $\Sigma$; ($\epsilon \in \Sigma^*$)
- $\Sigma^+$ is the set of all nonempty words over $\Sigma$ ($\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$)

**Definition**

A **formal language** $L$ is a set of words over an alphabet $\Sigma$, i.e. $L \subseteq \Sigma^*$. 
Type3-languages / right-linear languages

- the class of Type 3 languages can be generated by **right-linear grammars**

### Definition

A grammar \((N, T, S, R)\) is **Type3** or **right-linear** iff all rules are of the form:

\[
A \rightarrow a \text{ or } A \rightarrow wB \\
\text{with } A, B \in N, a \in T, \text{ and } w \in T^*
\]

Additionally, the rule \(S \rightarrow \epsilon\) is allowed iff \(S\) does not appear in any right-hand side of a rule.

A language generated by a right-linear grammar is said to be a **right-linear language** or a **Type3-language**.

[Remember, we write \(L(G)\) for the language generated by a grammar \(G\).]
Type3-languages / right-linear languages

Examples:

- \( P = \{ S \rightarrow aB, B \rightarrow bB, B \rightarrow bA, A \rightarrow a \} \)
  generates \((ab^*a)\)

- \( P = \{ S \rightarrow \epsilon, S \rightarrow aA, S \rightarrow bB, A \rightarrow aA, A \rightarrow \epsilon, A \rightarrow bB, B \rightarrow bB, B \rightarrow \epsilon \} \)
  generates \((a^*b^*)\)
Regular expressions

- the class of Type 3 languages can be described by **regular expressions**

The set of **regular expressions** $\text{RegEx}_\Sigma$ over an alphabet $\Sigma = \{a_1, \ldots, a_n\}$ is defined by:

- $\emptyset$ is a regular expression.
- $\epsilon$ is a regular expression.
- $a_1, \ldots, a_n$ are regular expressions.
- If $a$ and $b$ are regular expressions over $\Sigma$ then
  - $(a|b)$
  - $ab$
  - $a^*$

are regular expressions too.
Regular expressions

**RegEx: semantics**

Each regular expression \( r \) over an alphabet \( \Sigma \) denotes a formal language \( L(r) \subseteq \Sigma^* \).

*Regular languages* are those formal languages which can be expressed by a regular expression.

The denotation function \( L \) is defined inductively:

- \( L(\emptyset) = \emptyset \), \( L(\epsilon) = \{ \epsilon \} \), \( L(a_i) = \{ a_i \} \)
- \( L(r_1 | r_2) = L(r_1) \cup L(r_2) \)
- \( L(r_1 r_2) = L(r_1) \cdot L(r_2) \)
- \( L(r^*) = L(r)^* \)

‘\( r^+ \)’ is used as a short-hand for ‘\( r \cdot r^* \)’.
Examples: regular expressions

Find a regular expression which describes the regular language \( L \) (be careful: at least one language is not regular!)

- \( L \) is the language over the alphabet \( \{a, b\} \) with \( L = \{aa, \epsilon, ab, bb\} \).
  \[
  aa | \epsilon | ab | bb
  \]
- \( L \) is the language over the alphabet \( \{a, b\} \) which consists of all words which start with a nonempty string of \( a \)'s followed by any number of \( b \)'s. \( a^+ b^* \)
- \( L \) is the language over the alphabet \( \{a, b\} \) such that every \( a \) has a \( b \) immediately to the right. \( b^*(ab^+)^* \)
- \( L \) is the language over the alphabet \( \{a, b\} \) which consists of all words which contain an even number of \( a \)'s. \( b^*(ab^*a)^*b^* \)
- \( L \) is the language of all palindromes over the alphabet \( \{a, b\} \). **not regular!**
Deterministic finite-state automaton (detFSA)

- the class of Type 3 languages can be accepted (recognized) by **deterministic finite-state machines** (detFSA)
- example: detFSA for the language $L(a^+)$

\[
\begin{array}{c}
\text{start} \\ q_0 \\
\end{array} \quad \xrightarrow{a} \quad \begin{array}{c}
q_1 \\
\end{array}
\]

- initial state $q_0$, final state $q_1$
- transitions from $q_0$ to $q_1$ reading an $a$, from $q_1$ to $q_1$ reading an $a$
Deterministic finite-state automaton (detFSA)

Definition

A **deterministic finite-state automaton** is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) with:

1. a finite, nonempty set of **states** \(Q\)
2. an alphabet \(\Sigma\) with \(Q \cap \Sigma = \emptyset\)
3. a **transition function** \(\delta : Q \times \Sigma \rightarrow Q\)
4. an **initial state** \(q_0 \in Q\) and
5. a set of **final states** \(F \subseteq Q\)
Definition

A situation of a finite-state automaton \((Q, \Sigma, \delta, q_0, F)\) is a triple \((x, q, y)\) with \(x, y \in \Sigma^*\) and \(q \in Q\).

Situation \((x, q, y)\) produces situation \((x', q', y')\) in one step if there exists an \(a \in \Sigma\) such that \(x' = xa\), \(y = ay'\) and \(\delta(q, a) = q'\), we write \((x, q, y) \mapsto (x', q', y')\) \([ (x, q, y) \mapsto^* (x', q', y') \text{ as usual} ]\).

Definition

A word \(w \in \Sigma^*\) gets accepted by an automaton \((Q, \Sigma, \delta, q_0, F)\) if \((\epsilon, q_0, w) \mapsto^* (w, q_n, \epsilon)\) with \(q_n \in F\).

An automaton accepts a language iff it accepts every word of the language. We write \(L(A)\) for the language accepted by an automaton \(A\).
Examples (detFSA)

- accepting the language $L(ab^*c)$
  
  $M_1 = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{(q_0, a, q_1), (q_1, b, q_1), (q_1, c, q_2)\}, q_0, \{q_2\})$

  state diagram:

  ![State Diagram Example 1](image)

- accepting the language $a^*b^*$

  $M_1 = (\{q_0, q_1\}, \{a, b\}, \{(q_0, a, q_0), (q_0, b, q_1), (q_1, b, q_1)\}, q_0, \{q_0, q_1\})$
Example

- both automatons accept language $L((ab)^*)$
- in automaton graphs we often omit the trap state (partial transition function)
Examples

- accepting the language \((ha)^*\)!

```
start → 1 → h → 2 → a → 3 → ! → 4
```

```
start → 1 → h → 2 → a → 3 → ! → 4
```
Nondeterministic finite-state automaton (nondetFSA)

**Definition**

A *nondeterministic finite-state automaton* is a 5-tuple \((Q, \Sigma, \Delta, q_0, F)\) with:

1. *a finite nonempty set of states* \(Q\)
2. *an alphabet* \(\Sigma\) with \(Q \cap \Sigma = \emptyset\)
3. *a transition relation* \(\Delta \subseteq Q \times \Sigma \times Q\)
4. *an initial state* \(q_0 \in Q\) and
5. *a set of final states* \(F \subseteq Q\)

**nondetFSA: extensions**

- *an \(\varepsilon\)-transition* \(\xrightarrow{\varepsilon}\) allows to change the state without reading a symbol
- *a regular-expression transition* \(\xrightarrow{r}\) allows to change the state by reading in any string \(s \in L(r)\)
Equivalence of detFSA and nondetFSA

Theorem of Rabin & Scott

A language $L$ is accepted by a detFSA iff $L$ is accepted by a nondetFSA (with $\epsilon$-transitions and/or regular-expression transitions).

Why is it useful to have both notions?

- the detFSAs are conceptually more straightforward
- it is often easier to construct a nondetFSA
- for some other classes of automata the two subclasses are not equivalent, so the notions remain important

Example:

- $L : \{a^n \mid n \text{ is even or dividable by } 3\}$ (or $L((aa)^* \mid (aaa)^*)$)
- $L((aa)^* \mid (aaa)^*)$ is accepted by the automata on the following slides: regex-FSA, $\epsilon$-FSA, nondetFSA and detFSA
Equivalence of detFSA and nondetFSA

- \( L((aa)^* \mid (aaa)^*) \) with regex-FSA

\[
\begin{align*}
q_0 & \xrightarrow{\epsilon} q_1 \\
q_0 & \xrightarrow{\epsilon} q_2 \\
q_1 & \xrightarrow{aa} q_1 \\
q_2 & \xrightarrow{aaa} q_2
\end{align*}
\]
Equivalence of detFSA and nondetFSA

$L((aa)^* \mid (aaa)^*)$ with $\epsilon$-FSA

```
q0 -> a | q1
q1 -> a | q3
q3 -> a | q1

q0 -> \epsilon | q2
q2 -> a | q5
q5 -> a | q4
q4 -> a | q0
```

Petersen & Balogh (HHU)
Equivalence of detFSA and nondetFSA

- $L((aa)^* \mid (aaa)^*)$ with nondetFSA
Equivalence of detFSA and detFSA

\[ L((aa)^* \mid (aaa)^*) \] with detFSA
Eliminating $\varepsilon$-transitions

- The $\varepsilon$-closure of a state $q$ (denoted as $ECL(q)$) is the set that contains $q$ together with all states that can be reached starting at $q$ by following only $\varepsilon$-transitions.

- Given an $\varepsilon$-FSA $M$ eliminating $\varepsilon$-transitions produces an nondetFSA $M'$ such that $L(M') = L(M)$.

- The construction of $M'$ begins with $M$ as input, and takes 3 steps:
  1. Make $q$ an accepting state iff $ECL(q)$ contains an accepting state in $M$.
  2. Add an arc from $q$ to $q'$ labeled $a$ iff there is an arc labeled $a$ in $M$ from some state in $ECL(q)$ to $q'$.
  3. Delete all arcs labeled $\varepsilon$. 

Petersen & Balogh (HHU)  
Formal Language Theory  
NASSLLI 2014  
21 / 32
Eliminating $\epsilon$-transitions

- the $\epsilon$-FSA for $L((aa)^* \mid (aaa)^*)$

- $ECL(q_0) = \{q_0, q_1, q_2\}$
  1. make $q_0$ an accepting (final) state
  2. add the arcs: from $q_0$ to $q_3$ by $a$ and $q_0$ to $q_4$ by $a$
  3. Delete all arcs labeled $\epsilon$. 
Eliminating $\epsilon$-transitions

- step 1 – 2. resulting in:

- nondetFSA (see slide 19)
nondetFSA to detFSA

- make the nondetFSA from the previous slide deterministic
- remove multiple transitions with the same symbol
- idea: each state in detFSA will be a **set of states** from the nondetFSA
  - from $q_0$ we can go with $a$ to $q_3$ and $q_4$
    $\Rightarrow$ in the detFSA we have the states $\{q_0\}$ and $\{q_3, q_4\}$ with an $a$ transition

```
start → $\{q_0\}$ $\xrightarrow{a}$ $\{q_3, q_4\}$
```

- from the states in $\{q_3, q_4\}$ we can go with $a$ to $q_1$ and $q_5$
  $\Rightarrow$ in the detFSA we add the state $\{q_1, q_5\}$ with an $a$ transition from $\{q_3, q_4\}$

```
start → $\{q_0\}$ $\xrightarrow{a}$ $\{q_3, q_4\} \xrightarrow{a}$ $\{q_1, q_5\}$
```
nondetFSA to detFSA

- repeat the steps as before, result in in:

- make all states final, where any of the states in the set were final states in the nondetFSA
Theorem of Kleene

**Theorem**

*If* $L$ *is a formal language, the following statements are equivalent:*

- $L$ *is regular* (i.e., describable by a regular expression)
- $L$ *is right-linear* (i.e., generated by a right-linear grammar)
- $L$ *is FSA-acceptable* (i.e., accepted by a finite state automaton)

**Proof idea:**

1. every regular language is right-linear
2. every right-linear language is FSA-acceptable
3. every FSA-acceptable language is regular
Proof: Every regular language is right-linear

\[ \Sigma = \{ a_1, \ldots, a_n \} \]

1. \( L(\emptyset) \) is generated by \((\{S\}, \Sigma, S, \{\})\),
2. \( L(\epsilon) \) is generated by \((\{S\}, \Sigma, S, \{S \rightarrow \epsilon\})\),
3. \( L(a_i) \) is generated by \((\{S\}, \Sigma, S, \{S \rightarrow a_i\})\),
4. If \( L(r_1) \), \( L(r_2) \) are regular languages described by \( r_1 \), \( r_2 \) with generating right-linear grammars \((N_1, T_1, S_1, P_1)\), \((N_2, T_2, S_2, P_2)\), then \( L(r_1 \mid r_2) \) is generated by \((N_1 \cup N_2, T_1 \cup T_2, S, P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\})\),
5. \( L(r_1 r_2) \) is generated by \((N_1 \cup N_2, T_1 \cup T_2, S_1, P_1' \cup P_2)\) \((P_1' \) is obtained from \(P_1 \) if all rules of the form \(A \rightarrow b\) \((b \in T)\) are replaced by \(A \rightarrow bS_2\)),
6. \( L(r_1^*) \) is generated by \((N_1, \Sigma, S_1, P_1' \cup \{S_1 \rightarrow \epsilon, S_1 \rightarrow S\})\) \((P_1' \) is obtained from \(P_1 \) if by all rules of the form \(A \rightarrow b\) \((b \in T)\) we add the rule \(A \rightarrow bS\)).
Proof: Every right-linear language is FSA-acceptable

If $G = (N, T, S, R)$ is a right-linear grammar then the non-deterministic FSA $M = (N \cup \{\text{final}\}, T, \Delta, S, F)$ with

- $F = \{\text{final}, S\}$ if $S \rightarrow \epsilon \in R$ or else $F = \{\text{final}\}$.
- $(A, a, B) \in \Delta$, if $A \rightarrow aB \in R$ and $(A, a, \text{final}) \in \Delta$ if $A \rightarrow a \in R$.

accepts $L(G) = L(M)$.

$S \rightarrow aA, S \rightarrow bB, S \rightarrow \epsilon, A \rightarrow aA, A \rightarrow a, B \rightarrow bB, B \rightarrow b$
Every FSA-acceptable language is regular

Let $M = (Q, \Sigma, \Delta, q_0, F)$ be a nondetFSA.

1. Construct an equivalent automaton $M'$ with only one final state and no incoming transitions at the start state: $M' = (Q \cup \{q_s, q_f\}, \Sigma, \Delta', q_s, \{q_f\})$ with $\Delta' = \Delta \cup \{(q_s, \epsilon, q_0)\} \cup \{(q_i, \epsilon, q_f | q_i \in F)\}$.

2. For each pair of states $(q_i, q_j)$ replace all $(q_i, r_1, q_j) \in \Delta', (q_i, r_2, q_j) \in \Delta', \ldots$ by a single transition $(q_i, r_1 | r_2 | \ldots, q_j)$.

3. As long as there is still a state $q_k \notin \{q_s, q_f\}$ eliminate $q_k$ by the following rule:

4. Finally the automaton consists only of the two states $q_s$ and $q_f$ and one single transition $(q_s, r, q_f)$ and $L(M) = L(r)$. 
Example

- starting with the FSA:

- adding $\epsilon$-transitions:

- eliminating $q_1$:
  - $s \xrightarrow{a} q_2$
  - $s \xrightarrow{\epsilon} f$
  - $q_2 \xrightarrow{a} f$
  - $q_2 \xrightarrow{aa} q_2$
**Example**

- **starting with the FSA:**

  ![Finite-State Automaton](image1)

- **adding $\epsilon$-transitions:**

  ![Finite-State Automaton with $\epsilon$-transitions](image2)

- **eliminating $q_2$:**
  - $q_1 \xrightarrow{ab^*a} q_1$

  ![Finite-State Automaton after eliminating $q_2$](image3)
Intuitive rules for regular languages

- L is regular if it is possible to check the membership of a word simply by reading it symbol by symbol while using only a finite stack.
- Finite-state automatons are too weak for:
  - unlimited counting in \( \mathbb{N} \) ("same number as");
  - recognizing a pattern of arbitrary length ("palindrome");
  - expressions with brackets of arbitrary depth.