Formal Languages in Theory and Practice — day 2

Regular Languages

Wiebke Petersen & Kata Balogh

(Heinrich-Heine-Universität Düsseldorf)

ESSLLI 2019

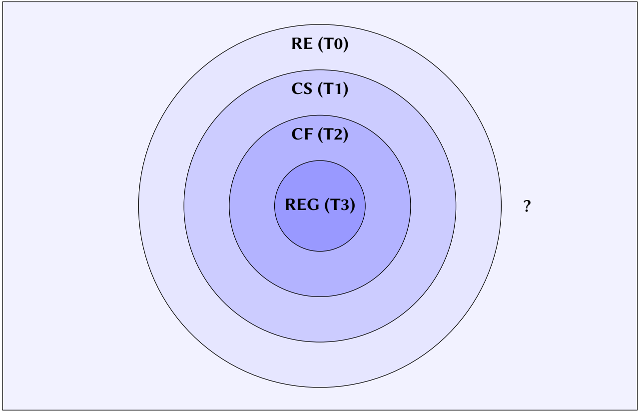
University of Latvia, Riga

Outline

Chomsky-hierarchy: main theorem

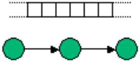
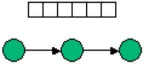
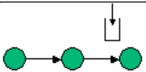
regular $\subset$ context-free $\subset$ context-sensitive $\subset$ recursively enumerable

REG $\subset$ CF $\subset$ CS $\subset$ RE



Recall

- **alphabet** Σ: nonempty, finite set of **symbols**
- **word** w: a finite string $x_1 \ldots x_n$ of symbols; $(x_1 \ldots x_n \in \Sigma)$
- a **formal language** L is a set of words over an alphabet Σ, i.e. $L \subseteq \Sigma^*$

type	grammar	rules	machine	idea	word problem
RE	unrestricted	$\alpha \to \beta$	Turing machine		undecidable
CS	context-sensitive	$\gamma A \delta \to \gamma \beta \delta$	linearly restricted automaton		exponential
CF	context-free	$A \to \beta$	pushdown-automaton		cubic
REG	right-linear	$A \to a \vert aB$	finite-state automaton		linear

Which is the class of natural languages?

Why is the formal complexity of natural languages interesting?

- It gives information about the general structure of natural language
- It allows to draw conclusions about the adequacy of grammar formalisms
- It determines a lower bound for the computational complexity of natural language processing tasks

Which is the class of natural languages?

Which idealizations about NL are necessary?

1. The family of natural languages exists.
2. Language = set of strings over an alphabet:
3. Natural languages are generated by finite rule systems (grammars)
4. Each NL consists of an *infinite* set of strings

About the idealizations

The family of natural languages exists:

- all natural languages are structurally similar
- all natural languages have a similar generative capacity

Arguments:

- all NLs serve for the same tasks
- children can learn each NL as their native language (within a similar period of time)

$\Rightarrow$ No evidence for a principal structural difference

About the idealizations (cont.)

Language = a set of strings over an alphabet:

- native speakers have full competence
- consistent grammaticality judgements

Arguments:

- all mistakes are due to performance not competence
- Mathews (1979) counter examples:
  - The canoe floated down the river sank.
  - The editor authors the newspaper hired liked laughed.
  - The man (that was) thrown down the stairs died.
  - The editor (whom) the authors the newspaper hired liked laughed.

About the idealizations (cont.)

Natural languages are generated by finite rule systems (grammars):

Arguments:

If a language is infinite, a finite set of rules can explain

- how a language can be learned
- how we understand each others sentences

About the idealizations (cont.)

Each NL consists of an *infinite* set of strings

Arguments:

- Recursion in NL:
  - John likes Peter
  - John likes Peter and Mary
  - John likes Peter and Mary and Sue
  - John likes Peter and Mary and Sue and Otto and …
- (Donaudampfschiffskapitänsmützenschirm …)

However:

- The set of all English sentences that have been used so far and that will be used in the time of mankind is finite.

Right-linear grammars

- 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:

$$A \rightarrow a \text{ or } A \rightarrow aB \text{ with } A, B \in N, a \in T$$

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.]

- **left-linear grammars** are defined analogously and generate Type 3 languages as well ($A \rightarrow a$ or $A \rightarrow Ba$)

Examples: Right-linear / left-linear grammar

generating the language (aaa^*)

with a right-linear grammar:

- $R = \{S \to aA, A \to aA, A \to a\}$
- example derivations:

  $S \Rightarrow aA \Rightarrow aa$

  $S \Rightarrow aA \Rightarrow aaA \Rightarrow aaaA \Rightarrow aaaa$

and with a left-linear grammar:

- $R = \{S \to Aa, A \to Aa, A \to a\}$
- example derivations:

  $S \Rightarrow Aa \Rightarrow aa$

  $S \Rightarrow Aa \Rightarrow Aaa \Rightarrow Aaaa \Rightarrow aaaa$

Regular expressions

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

The set of **regular expressions** *$RegEx_\Sigma$* over an alphabet $\Sigma = \{x_1, \ldots, x_n\}$ is defined by:

- $\emptyset$ is a regular expression.
- ϵ is a regular expression.
- $x_1, \ldots, x_n$ are regular expressions.
- If a and b are regular expressions over Σ then
  - $(a|b)$
  - ab
  - $a^\star$

  are regular expressions too.

Regular expressions

RegEx: semantics

Each regular expression r over an alphabet Σ 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:

- $L(\emptyset) = \emptyset$, $L(\epsilon) = \{\epsilon\}$, $L(x_i) = \{x_i\}$
- $L(r_1|r_2) = L(r_1) \cup L(r_2)$
- $L(r_1 r_2) = L(r_1) \frown L(r_2)$
- $L(r^*) = L(r)^*$

'r^+' is used as a short-hand for '$r \frown r^*$'.

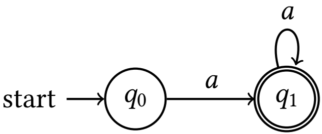
Examples: regular expressions

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 $L(a^+)$



- 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

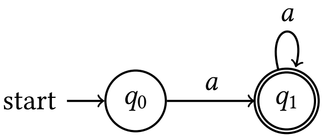
Deterministic finite-state automaton (detFSA)

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* Σ *with* $Q \cap \Sigma = \emptyset$
3. *a **transition function*** $\delta : Q \times \Sigma \to Q$
4. *an **initial state*** $q_0 \in Q$ *and*
5. *a set of **final states*** $F \subseteq Q$

$$FSA = (\{q_0, q_1\}, \{a\}, \{(q_0, a) \mapsto q_1, (q_1, a) \mapsto q_1\}, q_0, \{q_1\})$$



Language accepted by an automaton

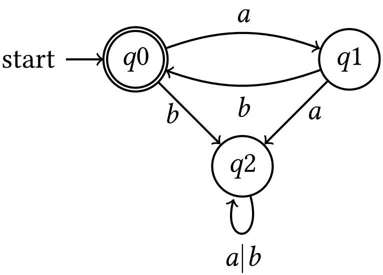
Definition

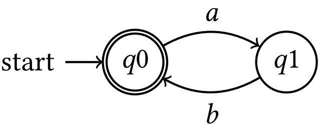
*A **situation** of a finite-state automaton* $(Q, \Sigma, \delta, q_0, F)$ *is a triple* (x, q, y) *with* $x, y \in \Sigma^*$ *and* $q \in Q$.
Situation (x, q, y) ***produces*** *situation* (x', q', y') ***in one step*** *if there exists an* $a \in \Sigma$ *such that* $x' = xa$, $y = ay'$ *and* $\delta(q, a) = q'$*, we write* $(x, q, y) \mapsto (x', q', y')$ *[*$(x, q, y) \mapsto^* (x', q', y')$ *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.

Example





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

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* Σ *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 **ϵ-transition** $\xrightarrow{\epsilon}$ 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

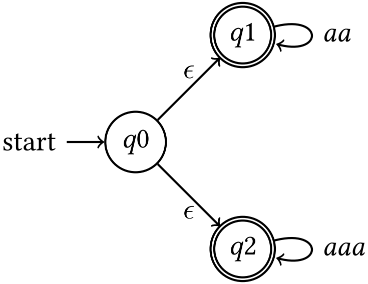
Theorem of Rabin & Scott

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

- Why is it useful to have both notions?
  - the detFSAs are conceptually more straightforward
  - sometimes easier to construct a nondetFSA
  - for some other classes of automata the two subclasses are not equivalent
- example:
  - $L : \{a^n \mid n$ is even or dividable by 3$\}$ (or $\mathrm{L}((aa)^* \mid (aaa)^*)$)
  - $\mathrm{L}((aa)^* \mid (aaa)^*)$ is accepted by the automata on the following slides: regex-FSA, ϵ-FSA, nondetFSA and detFSA

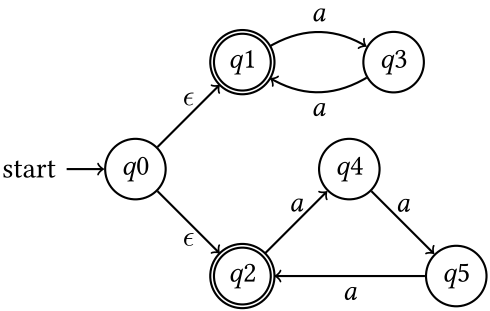
Equivalence of detFSA and nondetFSA

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



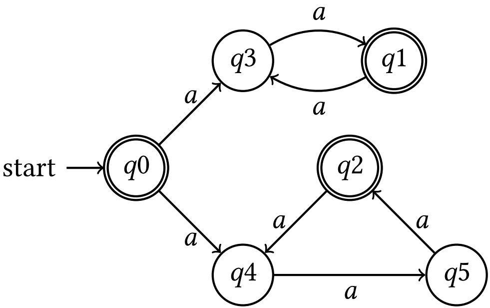
Equivalence of detFSA and nondetFSA

- L($(aa)^* \mid (aaa)^*$) with ϵ-FSA



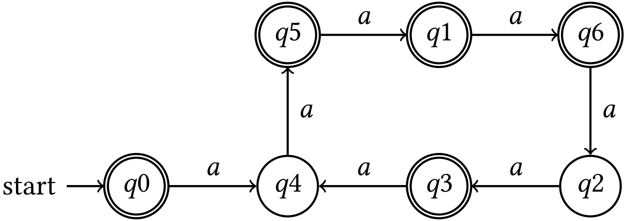
Equivalence of detFSA and nondetFSA

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



Equivalence of detFSA and detFSA

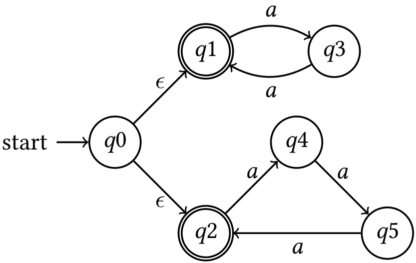
- L($(aa)^* \mid (aaa)^*$) with detFSA



Eliminating ϵ-transitions

- the ϵ-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 ϵ-transitions
- Given an ϵ-FSA M eliminating ϵ-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 ϵ.

Eliminating ϵ-transitions



- $ECL(q_0) = \{q_0, q_1, q_2\}$

1. make q_0 an accepting (final) state

   (Make q an accepting state iff $ECL(q)$ contains an accepting state in M.)

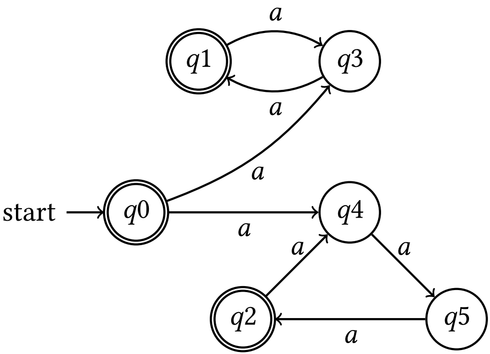
2. add the arcs: from q_0 to q_3 by a and q_0 to q_4 by a

   (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 ϵ.

Eliminating ϵ-transitions

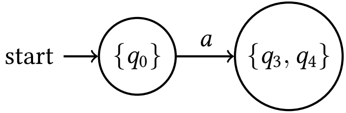
- step 1 – 2. resulting in:



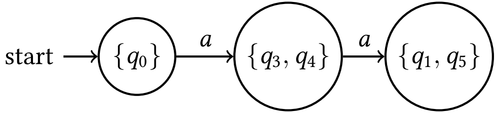
- still non-deterministic FSA

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

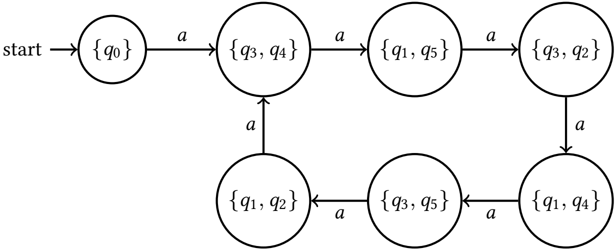


  - from the states in $\{q_3, q_4\}$ we can go with a to q_1 and q_5
    $\Rightarrow$ in the detFSA we add the state $\{q_1, q_5\}$ with an a transition from $\{q_3, q_4\}$

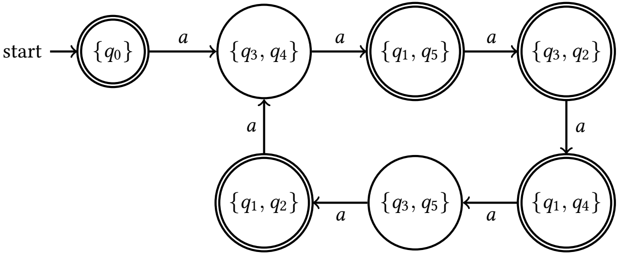


nondetFSA to detFSA

- 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 $(\{S\}, \Sigma, S, \{S \to \epsilon\})$,
3. $L(a_i)$ is generated by $(\{S\}, \Sigma, S, \{S \to a_i\})$,
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 $L(r_1|r_2)$ is generated by $(N_1 \uplus N_2, T_1 \cup T_2, S, P_1 \cup_{\uplus} P_2 \cup \{S \to S_1, S \to S_2\})$,
5. $L(r_1 r_2)$ is generated by $(N_1 \uplus N_2, T_1 \cup T_2, S_1, P_1' \cup_{\uplus} P_2)$ (P_1' is obtained from P_1 if all rules of the form $A \to b$ ($b \in T$) are replaced by $A \to bS_2$),
6. $L(r_1^*)$ is generated by $(N_1, \Sigma, S, P_1' \cup \{S \to \epsilon, S \to S_1\})$ (P_1' is obtained from P_1 if for all rules of the form $A \to b$ ($b \in T$) we add a rule $A \to bS_1$).

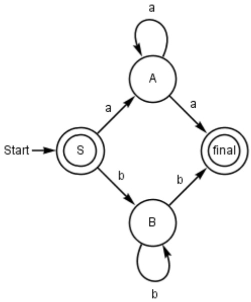
Proof: Every right-linear language is FSA-acceptable

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

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

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

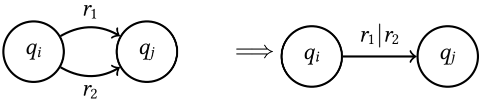
$$S \to aA,\ S \to bB,\ S \to \epsilon,\ A \to aA,\ A \to a,\ B \to bB,\ B \to b$$



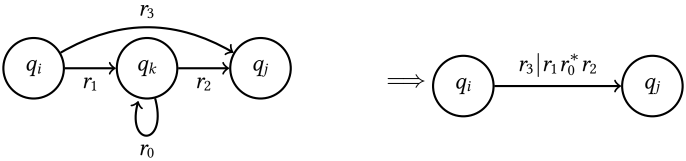
Every FSA-acceptable language is regular

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:

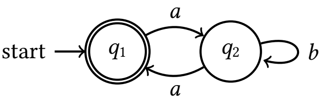


(Be careful, this last rule only illustrates the rough idea. To do it proper, you have to control the order in which you remove the states.)

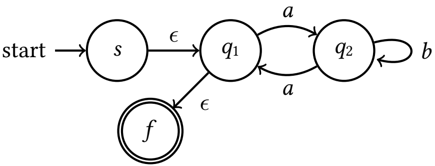
4. Finally the automaton consists only of the two states q_s and q_f and one single transition (q_s, r, q_f) and $L(M) = L(r)$.

Example

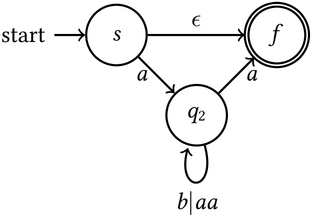
- starting with the FSA:

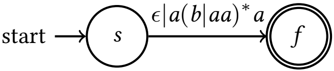


- adding ϵ-transitions:



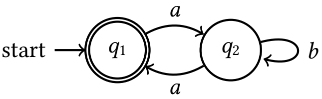
- eliminating q_1:
  - $s \xrightarrow{a} q_2$
  - $s \xrightarrow{\epsilon} f$
  - $q_2 \xrightarrow{a} f$
  - $q_2 \xrightarrow{b|aa} q_2$



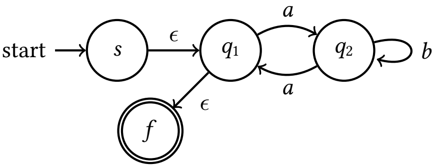


Example

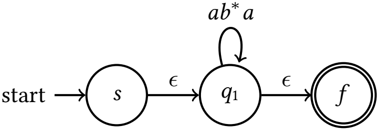
- starting with the FSA:

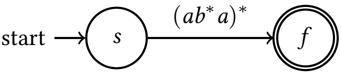


- adding ϵ-transitions:



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





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.

Closure properties of regular languages

A language class is closed under an operation if its application to arbitrary languages of this class results in a language of this class.

	Type3	Type2	Type1	Type0
union	+ ✓	+	+	+
intersection	+	-	+	+
complement	+	-	+	-
concatenation	+ ✓	+	+	+
Kleene's star	+ ✓	+	+	+
intersection with a regular language	+	+	+	+

complement: construct complementary DFSA

intersection: implied by *de Morgan*