# Lecture 2 – Myhill–Nerode theorem, Equivalent and Minimal Representations

NTIN071 Automata and Grammars

Jakub Bulín (KTIML MFF UK)

Spring 2024

** Adapted from the Czech-lecture slides by Marta Vomlelová with gratitude. The translation, some modifications, and all errors are mine.*

## Recap of Lecture 1

- Deterministic Finite Automaton (DFA): $A = (Q, \Sigma, \delta, q_0, F)$
- Extended transition function $\delta^*$
- The language recognized by the DFA $A$ is the language

$$L(A) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$$

- Languages recognized by some DFA are called regular
- Finite automata encode only finite information, but can recognize infinite languages
- Product automaton, intersection of reg. languages is regular
- Pumping lemma for regular languages (prove nonregularity)
- PL not a characterization, some nonregular can be pumped
- A regular language is infinite iff it contains a word of length $n \leq |w| \leq 2n$ where $n = \#$states of a recognizing automaton

# 1.4 Myhill–Nerode Theorem

## Characterize regular languages

How to recognize if a given language is regular? So far, we can construct a DFA recognizing the language, or use the Pumping lemma for contradiction.

We would like to have a characterization (which the Pumping Lemma is not!).

Luckily, as we'll see, every regular language comes with an implicit automaton 'hiding' in the set of all words over its alphabet.

## Congruences on words

Let $\Sigma$ be a finite alphabet and $\sim$ an equivalence relation on $\Sigma^*$ (reflexive, symmetric, transitive). Then:

- $\sim$ is a right congruence iff $(\forall u, v, w \in \Sigma^*) u \sim v \Rightarrow uw \sim vw$.
- $\sim$ has finite index iff the partition $\Sigma^* / \sim$ has a finite number of classes.
- the class containing a word $u$ is denoted $[u]_\sim$ or simply $[u]$

## The theorem

**Theorem (Mihyl–Nerode theorem)**

*Let $\Sigma$ be a finite alphabet and $L \subset \Sigma^*$ a language over $\Sigma$. The following statements are equivalent:*

(i) *$L$ is regular,*

(ii) *there exists a right congruence $\sim$ on $\Sigma^*$ with finite index such that $L$ is a union of some classes of the partition $\Sigma^* / \sim$.*

**Proof idea:** Group together words that end in the same state when we start reading from $q_0$.

## The proof

(i)$\Longrightarrow$(ii) from an automaton to a right congruence of finite index

- we define $u \sim v \equiv \delta^*(q_0, u) = \delta^*(q_0, v)$
- it is indeed a right congruence (from the definition of $\delta^*$)
- it has a finite index ($Q$ is finite)
- $L = \{w \mid \delta^*(q_0, w) \in F\} = \bigcup_{q \in F} \{w \mid \delta^*(q_0, w) = q\}$
  $= \bigcup_{q \in F} [w \mid \delta^*(q_0, w) = q]_\sim$ .

(ii)$\Longrightarrow$(i) from a right congruence of finite index to an automaton

- the alphabet is $\Sigma$, states $Q$ are the congruence classes $\Sigma^*/\sim$
- the initial state $q_0 = [\epsilon]$, final states $F = \{c_1, \ldots, c_n\}$ where
  $L = \bigcup_{i=1,\ldots,n} c_i$
- trans. function $\delta([u], x) = [ux]$ (well-defined right congruence)
- to show that $L(A) = L$, using $\delta^*([\epsilon], w) = [w]$:
  $w \in L \Leftrightarrow w \in \bigcup_{i=1,\ldots,n} c_i \Leftrightarrow w \in c_1 \vee \ldots w \in c_n \Leftrightarrow$
  $[w] = c_1 \vee \ldots [w] = c_n \Leftrightarrow [w] \in F \Leftrightarrow w \in L(A)$ $\square$

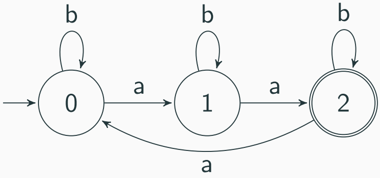
# Application: proof of regularity

**Example**

Construct an automaton that recognizes the language
$L = \{w \in \{a, b\}^* \mid |w|_a = 3k + 2 \text{ for some } k \geq 0\}$.

$$u \sim v \text{ iff } |u|_a \equiv |v|_a \pmod 3$$

- equivalence classes are 0,1,2
- $L$ corresponds to the class 2
- a – transitions to the next class
- b – stay in the same class.



**Example**

Show that $L = \{u \in \{a, b, c\}^* \mid u = a^+ b^i c^i \text{ or } u = b^i c^j\}$ is not regular. (Note that the first letter can be pumped.)

Suppose for contradiction that $L$ is regular. Let $\sim_L$ be a right congruence of finite index where $L$ is a union of some $\sim_L$-classes.

Consider the set of words $S = \{a, ab, abb, \ldots\} = \{ab^n \mid n \in \mathbb{N}\}$.

For any two $i \neq j$ there is a string $(c^i)$ distinguishing the words (in/out of the language): $ab^i c^i \in L$ but $ab^j c^i \notin L$

No two elements of $S$ can be in the same class of $\sim_L$ ($L$ would split the class). Since $S$ is infinite, this contradicts finite index of $\sim_L$. $\square$

# 1.5 Equivalent and Minimal Representations

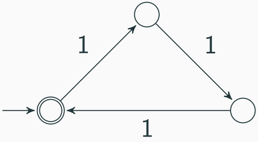
# Equivalent automata

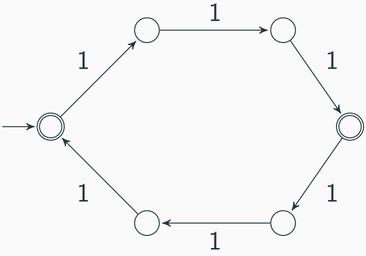
**Definition**

Finite automata $A, B$ are equivalent iff they recognize the same language, that is $L(A) = L(B)$.

**Example**

$L = \{w \in \{1\}^* \mid |w| = 3k \text{ for some } k \geq 0\}$



# Automata homomorphism

**Definition**

Let $A_1, A_2$ be DFAs. A surjective mapping $h : Q_1 \to Q_2$ is an (automata) homomorphism, if it satisfies:

- $h(\delta_1(q, x)) = \delta_2(h(q), x)$
- $h(q_{0_1}) = q_{0_2}$
- $q \in F_1 \Leftrightarrow h(q) \in F_2$

A bijective homomorphism is called an isomorphism.

(Isomorphic automata only differ by the 'names' of the states.)

**Theorem (Automata Equivalence Theorem)**

*Let $A_1, A_2$ be DFAs. If there exists a homomorphism from $A_1$ to $A_2$, then $A_1$ and $A_2$ are equivalent.*

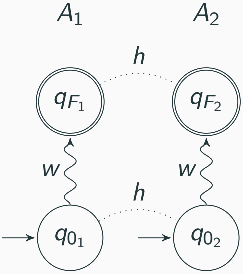
# The proof

For any $w \in \Sigma^*$, $q \in Q_1$, we can prove by finite induction that

$$h(\delta_1^*(q, w)) = \delta_2^*(h(q), w)$$

Then the following holds:

$$\begin{aligned} w \in L(A_1) &\Leftrightarrow \delta_1^*(q_{0_1}, w) \in F_1 \\ &\Leftrightarrow h(\delta_1^*(q_{0_1}, w)) \in F_2 \\ &\Leftrightarrow \delta_2^*(h(q_{0_1}), w) \in F_2 \\ &\Leftrightarrow \delta_2^*(q_{0_2}, w) \in F_2 \\ &\Leftrightarrow w \in L(A_2) \end{aligned}$$




□

# Reducing automata

The smallest DFA recognizing a given language?

Start with any DFA.

Two steps:

- remove unreachable states
- merge equivalent (indistinguishable) states

The reduced DFA is unique (up to automata isomorphism).

# (Un)reachable states

**Definition (Reachable states)**

Let's have a DFA $A = (Q, \Sigma, \delta, q_0, F)$ and $q \in Q$. The state $q$ is reachable iff there exists $w \in \Sigma^*$ such that $\delta^*(q_0, w) = q$.

**Algorithm (Reachable States – BFS on the state diagram)**

- *set* $M_0 = \{q_0\}$
- *repeat* $M_{i+1} = M_i \cup \{q \in Q \mid (\exists p \in M_i, \exists x \in \Sigma)\ \delta(p, x) = q\}$
- *until* $M_{i+1} = M_i$
- *return* $M_i$

**Proof of correctness and completeness.**

Corectness: $M_0 \subseteq M_1 \subseteq \ldots \subseteq Q$ and consist of reachable states.
Completeness: Let $q$ be reachable. Let $w = x_1 \ldots x_n$ be shortest such that $\delta^*(q_0, x_1 \ldots x_n) = q$. As $\delta^*(q_0, x_1 \ldots x_i) \in M_i \setminus M_{i-1}$ we get $\delta^*(q_0, x_1 \ldots x_n) = q \in M_n$. $\square$

# (In)distinguishable states

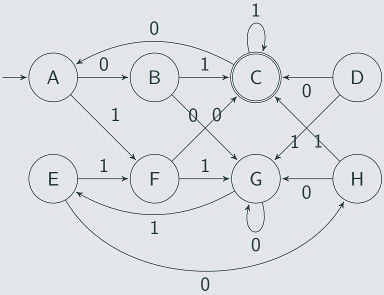
**Definition (State equivalence)**

States $p, q \in Q$ of a DFA $A$ are equivalent (indistinguishable), if for all words $w$: $\delta^*(p, w) \in F \Leftrightarrow \delta^*(q, w) \in F$

**Observation**

*State equivalence is indeed reflexive, symmetric and transitive.*

**Example**




C, G distinguishable, $\delta^*(C, \epsilon) \in F$, $\delta^*(G, \epsilon) \notin F$

A,G too: $\delta^*(A, 01) = C$ accepting, $\delta^*(G, 01) = E$ not accepting.

A,E equivalent (for $\epsilon$, $1*$ obvious, 0 goes to non-accepting, 01 & 00 meet in the same state)

# Recognizing state equivalence

Distinguish accepting from nonaccepting. Then go backwards.

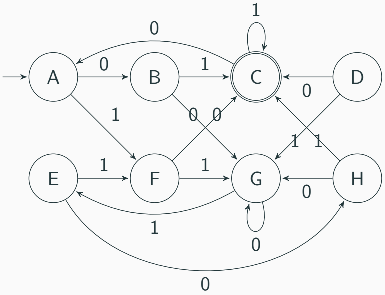
**Algorithm (Finding distinguishable states in a DFA)**

- *Basis: If $p \in F$ is accepting and $q \notin F$ is not, the pair $\{p, q\}$ is distinguishable.*
- *Induction: Let $p, q \in Q$ and $a \in \Sigma$. If $r = \delta(p, a)$ and $s = \delta(q, a)$ are distinguishable, then so are $p$ and $q$. (Repeat until no newly distinguished pair.)*

# Example 1/4

1. Accepting vs. non–accepting

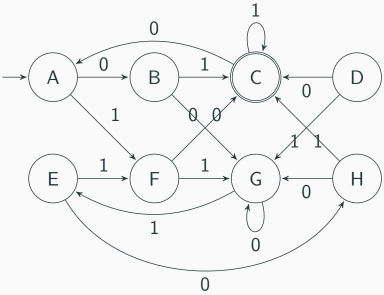
| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| B | | | | | | | |
| C | × | × | | | | | |
| D | | | × | | | | |
| E | | | × | | | | |
| F | | | × | | | | |
| G | | | × | | | | |
| H | | | × | | | | |



# Example 2/4

2. $\delta(q, 1) \in \mathcal{F}$ for $q \in \{B, C, H\}$

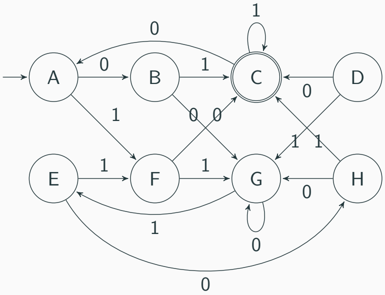
| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| B | $\times$ | | | | | | |
| C | $\times$ | $\times$ | | | | | |
| D | | $\times$ | $\times$ | | | | |
| E | | $\times$ | $\times$ | | | | |
| F | | $\times$ | $\times$ | | | | |
| G | | $\times$ | $\times$ | | | | |
| H | $\times$ | | $\times$ | $\times$ | $\times$ | $\times$ | $\times$ |



# Example 3/4

3. $\delta(q, 0) \in \mathcal{F}$ for $q \in \{D, F\}$

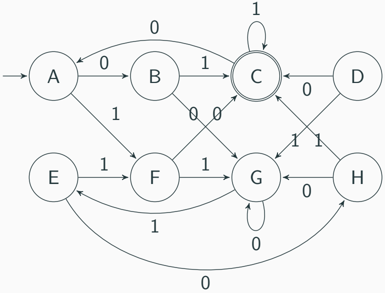
| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| B | $\times$ | | | | | | |
| C | $\times$ | $\times$ | | | | | |
| D | $\times$ | $\times$ | $\times$ | | | | |
| E | | $\times$ | $\times$ | $\times$ | | | |
| F | $\times$ | $\times$ | $\times$ | | $\times$ | | |
| G | | $\times$ | $\times$ | $\times$ | | $\times$ | |
| H | $\times$ | | $\times$ | $\times$ | $\times$ | $\times$ | $\times$ |



# Example 4/4

4. $B$ and $G$ are distinguishable, $\delta(A, 0) = B$, $\delta(G, 0) = G$, therefore $A$, $G$ are distinguishable. Similarly, $\delta(*, 0)$ for $E, G$ goes to distinguishable states $H, G$.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| B | × | | | | | | |
| C | × | × | | | | | |
| D | × | × | × | | | | |
| E | | × | × | × | | | |
| F | × | × | × | | × | | |
| G | × | × | × | × | × | × | |
| H | × | | × | × | × | × | × |
| | A | B | C | D | E | F | G |




Equivalent pairs:

$(A, E)$, $(B, H)$, $(D, F)$

# Correctness

**Theorem**

*A pair of states is not distinguished by the algorithm, if and only if the states are equivalent.*

**Proof.**

$\Leftarrow$ Clearly, only distinguishable pairs are distinguished.

$\Rightarrow$ Induction on the length of a shortest distinguishing word. If $p, q$ are distinguished by $w = \epsilon$, then the algorithm distinguishes them. Now let $w = a_1 \ldots a_k$. By induction, $r = \delta(p, a_1)$ and $s = \delta(q, a_1)$ are distinguished by the algorithm. But then the algorithm distinguishes $p, q$ in the next round (following $a_1$-transitions backwards). $\square$

## Complexity

The time complexity is polynomial in the number of states $n$.

- In one iteration, we consider all pairs, that is $O(n^2)$.
- In each iteration we add a cross, that means no more than $O(n^2)$ iterations.
- Together, $O(n^4)$.

The algorithm may be sped up to $O(n^2)$ by memorizing states that depend on the pair $\{r, s\}$ and following the list backwards.

**Exercise**

- Describe the $O(n^2)$ algorithm hinted above.
- The algorithm can also compute, for each distinguishable pair, the shortest word distinguishing that pair.

# Application: testing equality of regular languages

- regular languages $L, M$ are given by some representations
- from those construct DFA $A_L, A_M$ recognizing $L, M$
- we can assume $Q_L \cap Q_L = \emptyset$ (otherwise rename the states)
- run the following algorithm:

**Algorithm (Testing equivalence of automata)**

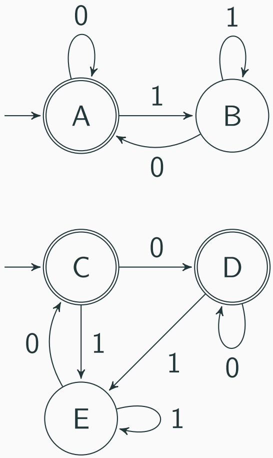
- *Construct a DFA* $B = (Q_L \cup Q_M, \Sigma, \delta_L \cup \delta_M, q_L, F_L \cup F_M)$ *as a union of states and transitions; select one (any) initial state.*
- *Test if* ${q_0}_L$ *and* ${q_0}_M$ *are equivalent. (The automata are equivalent iff their initial states are equivalent in* $B$*.)*

**NB:** Renaming states gives an isomorphic (hence equivalent) automaton. So we can always assume disjoint states. Alternatively, we can use disjoint union: $Q_B = Q_L \dot\sqcup Q_M = Q_L \times \{0\} \cup Q_M \times \{1\}$.

# Example

## Example

Two different DFAs recognizing $L = \{\epsilon\} \cup \{w0 \mid w \in \{0,1\}^*\}$.




| | A | B | C | D |
|---|---|---|---|---|
| B | $\times$ | | | |
| C | | $\times$ | | |
| D | | $\times$ | | |
| E | $\times$ | | $\times$ | $\times$ |

# Reduced automaton

## Definition (Reduced DFA)

A DFA $A$ is reduced iff all states are reachable, no two distinct states are equivalent, and there is no 'fail' state from which no accepting state would be reachable.

It is a reduct of a DFA $B$ iff it is reduced and equivalent to $B$.

## Theorem (DFA minimization)

(i) *Any two equivalent reduced automata are isomorphic.*

(ii) *Any DFA accepting at least one word has a reduct, unique up to automata isomorphism.*

## Proof.

(i) Any $q \in Q_1$ is reachable. Find a word $w$ s.t. $q = \delta_1^*(q_{0_1}, w)$. Define $h(q) = \delta_2^*(q_{0_2}, w)$. Check that $h$ is an isomorphism.
(ii) The construction described on the next slide works. $\square$

# Algorithm: constructing the reduct

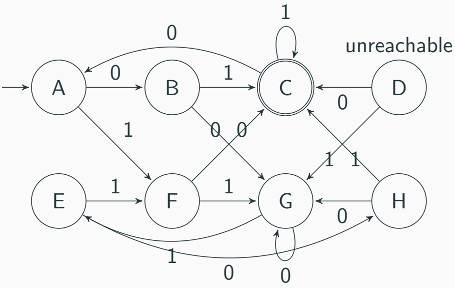
**input:** a DFA $A$

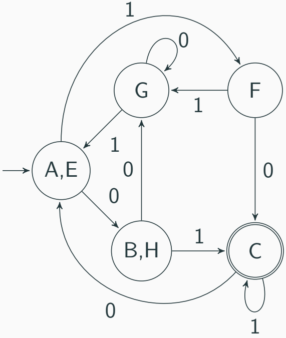
**output:** a DFA $B$ which is the reduct of $A$

1. eliminate from $A$ all unreachable states
2. find the indistinguishability partition on the remaining states
3. construct the reduct $B$:

- $Q_B$ are the equivalence classes
- $q_{0_B}$ is the class containing the initial state of $A$
- final states $F_B$ are the classes containing some state from $F_A$
- the transition function: for any $a \in \Sigma$ and $S \in Q_B$ choose arbitrary $q \in S$ and define $\delta_B(S, a) = [\delta_A(q, a)]$, i.e., the class containing $\delta_A(q, a) \in Q_A$; note that this class is the same for any choice of $q \in S$ since they are all equivalent
- if there's a 'fail' state from which no final states can be reached [and if we allow partial transition functions], remove it

# Example




| | | | | | | |
|---|---|---|---|---|---|---|
| B | × | | | | | |
| C | × | × | | | | |
| E | | × | × | | | |
| F | × | × | × | × | | |
| G | × | × | × | × | × | |
| H | × | | × | × | × | × |
| | A | B | C | E | F | G |

Equivalence classes:

$$\{A, E\}, \{B, H\}, \{C\}, \{F\}, \{G\}$$

# Summary of Lecture 2

- Mihyll–Nerode theorem (DFAs $\leftrightarrow$ right congruences of $\Sigma^*$ of finite index where $L$ is a union of classes)
- Equivalent automata (recognize the same language), automata homomorphism (implies automata equivalence).
- Finding reachable states: BFS on the state diagram
- Finding equivalent (indistinguishable) states: a table-filling algorithm
- Testing equivalence of DFAs, equality of regular languages
- Reduced (minimum–state) DFA, an algorithm to reduce a given DFA (using the equivalent states algorithm)