



INF 332: LANGUAGES & AUTOMATA

Chapter 7: Non-deterministic Automata

Yliès Falcone

ylies.falcone@univ-grenoble-alpes.fr — www.ylies.fr

Univ. Grenoble-Alpes
—
Laboratoire d'Informatique de Grenoble - www.liglab.fr

Outline Chapter 7: Non-deterministic Automata

- 1 Non-deterministic Finite-state Automata
- 2 Determinizing non-Deterministic Automata
- 3 Applications in Computer Science
- 4 Summary

Outline Chapter 7: Non-deterministic Automata

- 1 Non-deterministic Finite-state Automata
 - Idea and Motivation
 - Definition and Recognized Language
- 2 Determinizing non-Deterministic Automata
- 3 Applications in Computer Science
- 4 Summary

Idea and Motivation

Idea

- **Determinism:** For each state and each alphabet symbol, there is at most one successor state (0 or 1).
- **Nondeterminism:** For a state and a symbol, there may be 0, 1, or several successor states.

Motivations

- It is often easier to design a nondeterministic automaton that recognizes a language L than a deterministic one.
- For some languages, a nondeterministic automaton can be smaller than any deterministic automaton recognizing the same language.

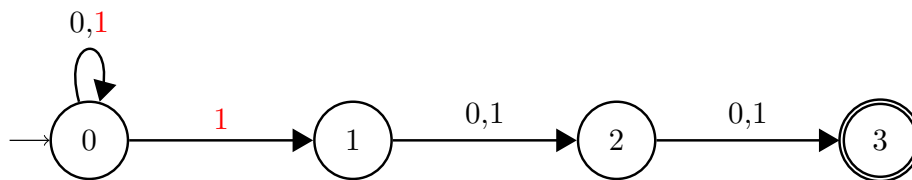
However

We will see that we cannot dispense with deterministic automata.

NFAs: Example

Let $\Sigma = \{0, 1\}$.

Let L_3 be the language of words of length ≥ 3 whose 3rd symbol from the right is 1.



The smallest deterministic automaton recognizing L_3 has 8 states.

More generally

Let L_k be the language of words of length $\geq k$ whose k^{th} symbol from the right is 1. No deterministic automaton with fewer than 2^k states can recognize L_k .

Outline Chapter 7: Non-deterministic Automata

- 1 Non-deterministic Finite-state Automata
 - Idea and Motivation
 - Definition and Recognized Language
- 2 Determinizing non-Deterministic Automata
- 3 Applications in Computer Science
- 4 Summary

NFAs

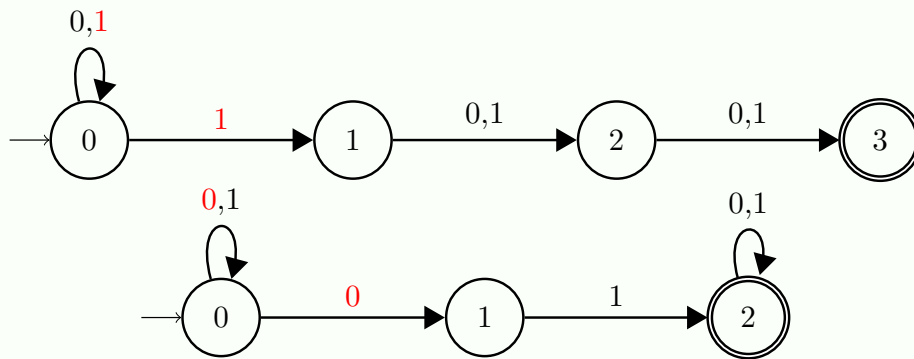
Definition

Definition (Nondeterministic Finite Automaton (NFA))

An NFA is defined by a 5-tuple $(Q, \Sigma, q_{\text{init}}, \Delta, F)$ where:

- Q is a finite set of **states**,
- Σ is the input alphabet,
- $q_{\text{init}} \in Q$ is the **initial state**,
- $\Delta \subseteq Q \times \Sigma \times Q$ is the **transition relation**,
- $F \subseteq Q$ is the set of **accepting states**.

Example (NFA)



Nondeterministic Finite Automata

Configuration, Derivation, Runs

Let $A = (Q, \Sigma, q_{\text{init}}, \Delta, F)$ be an NFA.

Definition (Configuration)

A **configuration** of A is a pair (q, u) where $q \in Q$ and $u \in \Sigma^*$.

Definition (Derivation relation)

The relation $\rightarrow_{\Delta}$ of **derivation** between configurations is defined by:

$$\forall q \in Q, \forall a \in \Sigma, \forall u \in \Sigma^* : (q, a \cdot u) \rightarrow_{\Delta} (q', u) \text{ iff } (q, a, q') \in \Delta.$$

Definition (Run)

A **run of A** on input u is a sequence of configurations $(q_0, u_0) \cdots (q_n, u_n)$ such that

$$\forall i \in \{0, \dots, n-1\} : (q_i, u_i) \rightarrow_{\Delta} (q_{i+1}, u_{i+1}).$$

- $u_0 = u,$
- $u_n = \epsilon,$
- $q_0 = q_{\text{init}}.$

We write $\xrightarrow{*}_{\Delta}$ for the reflexive and transitive closure of $\rightarrow_{\Delta}$.

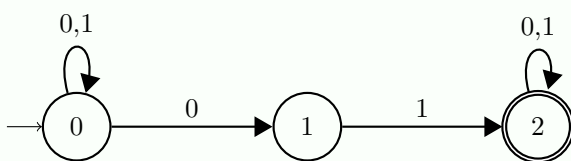
Acceptance of a Word by an NFA

Let $A = (Q, \Sigma, q_{\text{init}}, \Delta, F)$ be an NFA.

Definition (Word acceptance)

A word $u \in \Sigma^*$ is **accepted** by A if there exists a run of A on u such that the state in its final configuration is an accepting state.

Example (Word acceptance by an NFA)



Accepted words:

- 01 via the run $(0, 01) \cdot (1, 1) \cdot (2, \epsilon)$
- 001 via the run $(0, 001) \cdot (0, 01) \cdot (1, 1) \cdot (2, \epsilon)$

Language Recognized by an NFA

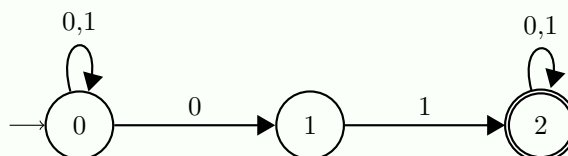
Definition (Recognized language)

The **language recognized by A** , denoted $L(A)$, is

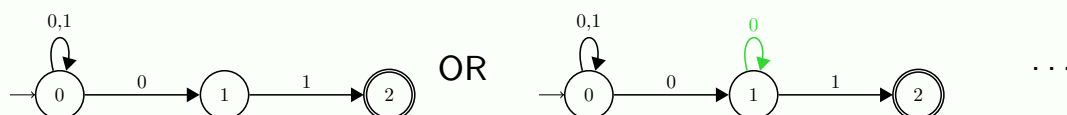
$$\{u \in \Sigma^* \mid u \text{ is accepted by } A\}.$$

Example (Recognized language)

- Words over $\Sigma = \{0, 1\}$ that contain a 0 followed by a 1:



- Words over $\Sigma = \{0, 1\}$ that end with a 0 followed by a 1:



NFAs vs DFAs

Using NFAs often simplifies the design of an automaton recognizing/defining a language.

We obviously have:

Every DFA is an NFA

By definition.

We will now show:

Every NFA has an equivalent DFA

By determinization (the subset construction method).

Outline Chapter 7: Non-deterministic Automata

- 1 Non-deterministic Finite-state Automata
- 2 Determinizing non-Deterministic Automata
 - Determinization Procedure
 - Correctness of the Determinization Procedure
 - On the Complexity of Determinization
- 3 Applications in Computer Science
- 4 Summary

Determinization Procedure (Subset Construction)

The Idea

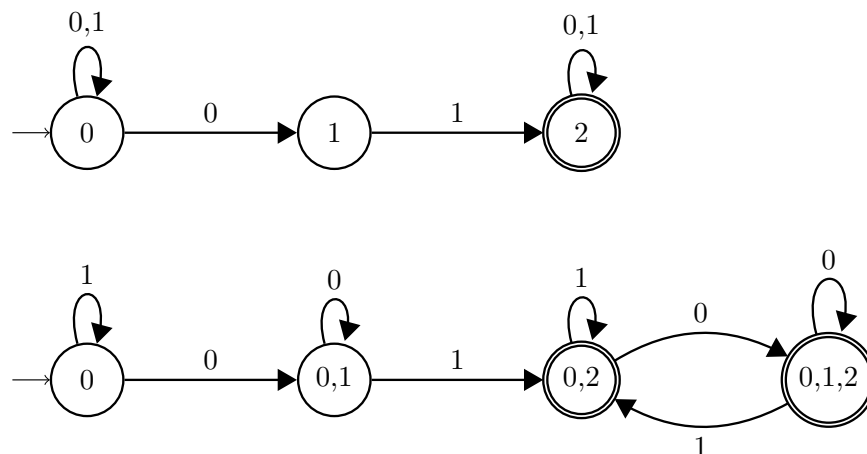
Goal of the procedure:

- Input: an NFA,
- Output: a DFA that recognizes the same language as the input automaton.

Rabin & Scott (1959)

In the deterministic automaton, a state reachable by a word u encodes *all* the states that can be reached by u in the nondeterministic automaton.

Let $\Sigma = \{0, 1\}$.



Determinization Procedure

Let $A = (Q, \Sigma, q_{\text{init}}, \Delta, F)$ be an NFA.

Definition (Determinized automaton)

The **determinization** of A , denoted $\text{Det}(A)$, is the deterministic finite automaton:

$$(\mathcal{P}(Q), \Sigma, \{q_{\text{init}}\}, \delta, \mathcal{F})$$

where:

- $\delta(X, a) = \{q' \mid \exists q \in X : (q, a, q') \in \Delta\}$,
- $\mathcal{F} = \{X \in \mathcal{P}(Q) \mid X \cap F \neq \emptyset\}$ (i.e., X is accepting iff X contains at least one accepting state).

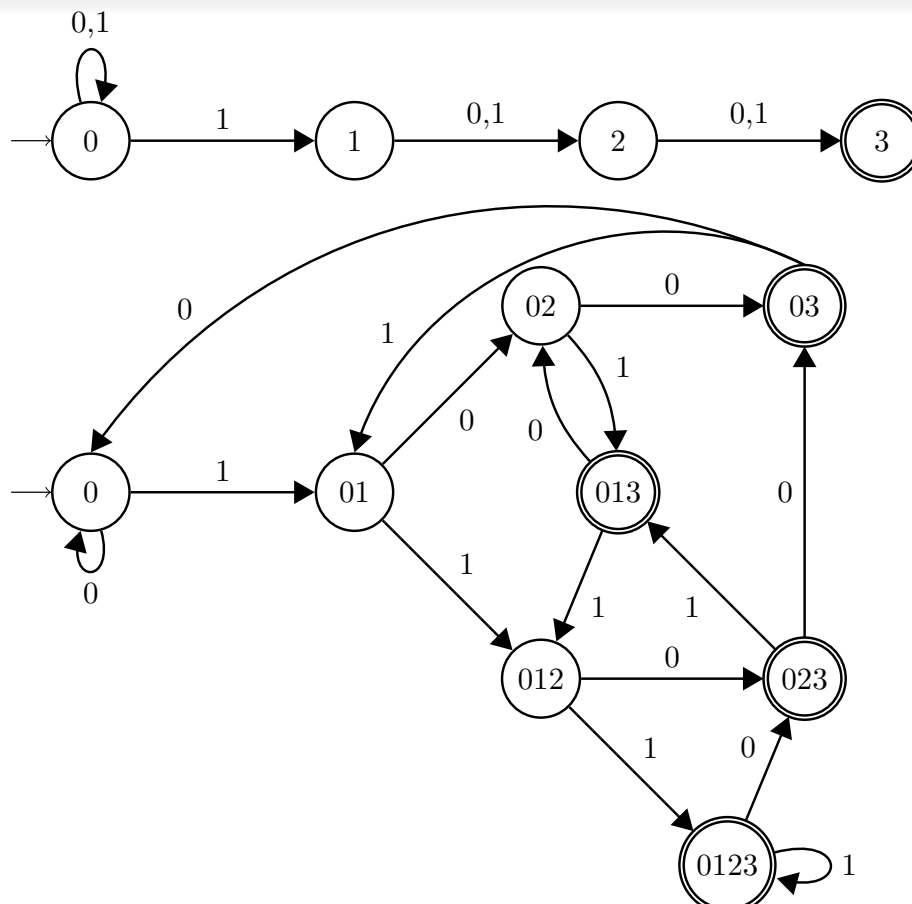
Intuition

- The set of states of the determinized automaton ($\mathcal{P}(Q)$) is the power set of Q .
- From a set of states $X \subseteq Q$, the transition on a symbol is the set of states reachable from X on that symbol.
- Accepting states are those subsets that contain at least one accepting state.

Remark In practice, only the *reachable subsets* of Q are needed. The full power set $\mathcal{P}(Q)$ defines the state space, but many subsets are unreachable and can be ignored. $\square$

Determinization Procedure

Example



Outline Chapter 7: Non-deterministic Automata

- 1 Non-deterministic Finite-state Automata
- 2 Determinizing non-Deterministic Automata
 - Determinization Procedure
 - Correctness of the Determinization Procedure
 - On the Complexity of Determinization
- 3 Applications in Computer Science
- 4 Summary

Determinization Procedure

Correctness of the Procedure

Recall: for $\delta \subseteq Q \times \Sigma \times Q$ a transition relation (which may be a function), δ^* denotes the reflexive and transitive closure of δ .

Extending the transition function

Represent the set of states reached from a set of states:

- on a single symbol: $\delta : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q)$,
- on a word: $\delta^* : \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$.

from	deterministic automaton (transition function δ)	nondeterministic automaton (transition relation Δ)
a symbol	$\delta(Q, a) = \bigcup_{q \in Q} \{\delta(q, a)\}$	$\delta(Q, a) = \bigcup_{q \in Q} \{q' \mid (q, a, q') \in \Delta\}$
a word	$\delta^*(Q, \epsilon) = Q$ $\delta^*(Q, x \cdot a) = \delta(\delta^*(Q, x), a)$	

Determinization Procedure

Correctness of the Procedure

Theorem: Correctness of determinization

$$L(\text{Det}(A)) = L(A)$$

Proof

Let $D = (Q^D, \Sigma, \{q_{\text{init}}\}, \delta_D, F^D)$ be a DFA obtained by determinizing the NFA $N = (Q^N, \Sigma, q_{\text{init}}, \delta_N, F^N)$.

- Prove $\delta_D^*(\{q_{\text{init}}\}, w) = \delta_N^*(q_{\text{init}}, w)$ by induction on w (i.e., structural induction on word length).
- D and N both accept $w \in \Sigma^*$ iff $\delta_D^*(\{q_{\text{init}}\}, w) \cap F^N \neq \emptyset$ and $\delta_N^*(q_{\text{init}}, w) \cap F^N \neq \emptyset$, respectively.

Determinization Procedure

Proof of Correctness

Proof of $\delta_D^*(\{q_{\text{init}}\}, w) = \delta_N^*(q_{\text{init}}, w)$ by induction on w

Base $|w| = 0$, i.e., $w = \epsilon$. By the definitions of transition functions for DFAs and NFAs:

$$\delta_D^*(\{q_{\text{init}}\}, \epsilon) = \delta_N^*(q_{\text{init}}, \epsilon) = \{q_{\text{init}}\}.$$

Induction step Let $w = x \cdot a$ with $x \in \Sigma^*$ and $a \in \Sigma$, and assume the hypothesis holds for x .

- By the induction hypothesis, $\delta_D^*(\{q_{\text{init}}\}, x) = \delta_N^*(q_{\text{init}}, x) \subseteq Q^N$.
- Write this set as $\{p_1, p_2, \dots, p_k\}$.
- By the inductive definition of δ_N^* :

$$\delta_N^*(q_{\text{init}}, w) = \bigcup_{i=1}^k \delta_N(p_i, a). \quad (\text{Eq.1})$$

- By the definition of determinization:

$$\delta_D(\{p_1, p_2, \dots, p_k\}, a) = \bigcup_{i=1}^k \delta_N(p_i, a). \quad (\text{Eq.2})$$

- Using (Eq.2) and $\delta_D^*(\{q_{\text{init}}\}, x) = \{p_1, \dots, p_k\}$ and the inductive definition of δ_D^* for DFAs:

$$\begin{aligned} \delta_D^*(\{q_{\text{init}}\}, w) &= \delta_D(\delta_D^*(\{q_{\text{init}}\}, x), a) \\ &= \delta_D(\{p_1, p_2, \dots, p_k\}, a) \\ &= \bigcup_{i=1}^k \delta_N(p_i, a). \quad (\text{Eq.3}) \end{aligned}$$

Outline Chapter 7: Non-deterministic Automata

- 1 Non-deterministic Finite-state Automata
- 2 Determinizing non-Deterministic Automata
 - Determinization Procedure
 - Correctness of the Determinization Procedure
 - On the Complexity of Determinization
- 3 Applications in Computer Science
- 4 Summary

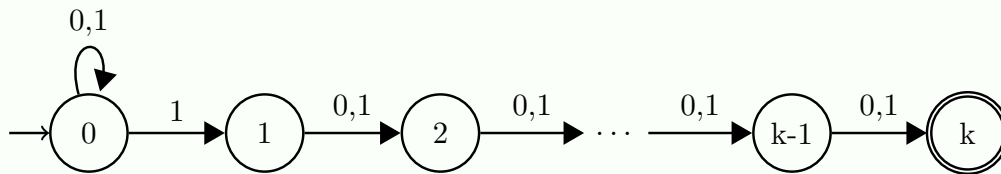
On the Complexity of Determinization

In practice, the size of the DFA produced is often comparable to that of the original NFA.

Example (A bad-case scenario)

Let $\Sigma = \{0, 1\}$ and let L_k be the language of words of length $\geq k$ where the k^{th} symbol from the right is 1:

$$L_k = \{a_1 \cdots a_n \mid n \geq k \wedge a_{n-k+1} = 1\}.$$



On the Complexity of Determinization

This Example

Lemma

No deterministic automaton with fewer than 2^k states recognizes L_k .

Proof (by contradiction)

- Let $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ be a deterministic automaton with $|Q| < 2^k$ and $L(A) = L_k$.
- Let $u = a_1 \cdots a_k$ and $v = b_1 \cdots b_k$ be two distinct words of length k such that $\delta^*(q_{\text{init}}, u) = \delta^*(q_{\text{init}}, v)$. Such words exist because there are 2^k words of length k but only $|Q| < 2^k$ states. Let $q = \delta^*(q_{\text{init}}, u)$.
- Since u and v differ, there exists i such that $a_i \neq b_i$. WLOG (by symmetry of $\neq$), assume $a_i = 1$ and $b_i = 0$.
- Define $u' = u0^{i-1}$ and $v' = v0^{i-1}$. Then:
 - $u'(|u'| - k + 1) = u'(k + i - 1 - k + 1) = u'(i) = a_i = 1$,
 - $v'(|v'| - k + 1) = v'(i) = b_i = 0$.

Hence $u' \in L_k$ and $v' \notin L_k$. This implies $\delta^*(q, 0^{i-1}) \in F$ and $\delta^*(q, 0^{i-1}) \notin F$ — a contradiction.

- Therefore, $\delta^*(q_{\text{init}}, u) = \delta^*(q_{\text{init}}, v)$ is impossible.

Outline Chapter 7: Non-deterministic Automata

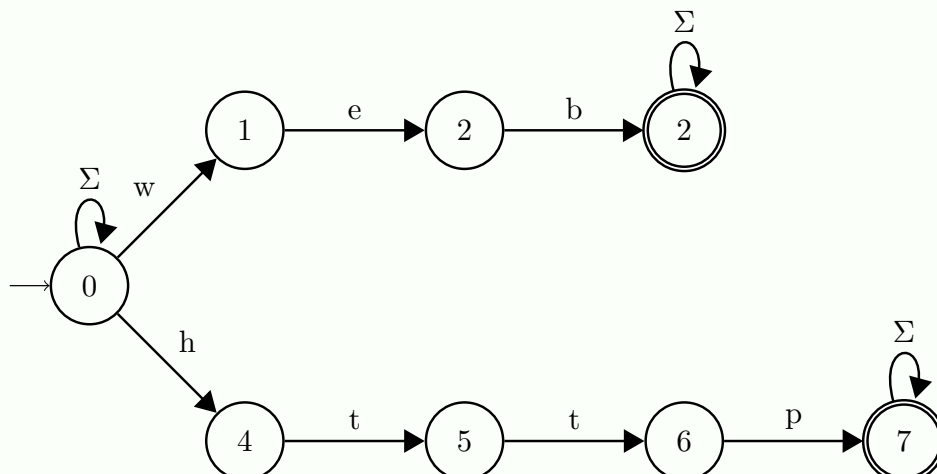
- 1 Non-deterministic Finite-state Automata
- 2 Determinizing non-Deterministic Automata
- 3 Applications in Computer Science
- 4 Summary

Applications in Computer Science

Several applications in computer science:

- Text recognition (e.g., in web browsers).
- Compilation: lexical analysis (recognition of programming language keywords).
- System specification: nondeterminism used to model unknown aspects (the environment).

Example (Recognizing a set of keywords)



Outline Chapter 7: Non-deterministic Automata

- 1 Non-deterministic Finite-state Automata
- 2 Determinizing non-Deterministic Automata
- 3 Applications in Computer Science
- 4 Summary

Outline Chapter 7: Non-deterministic Automata

Summary

Nondeterministic Finite Automata

- Definition
- Acceptance criterion, recognized language
- NFAs vs DFAs: conciseness
- Determinization procedure
 - algorithm
 - correctness
 - complexity insight
- Applications of NFAs in computer science