Advanced Logic

http://www-sop.inria.fr/members/Martin.Avanzini/teaching/2022/AL/

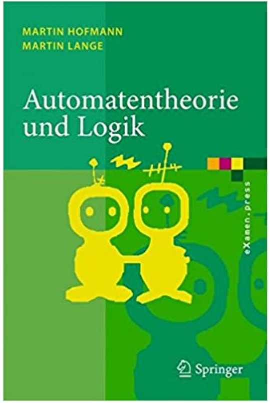
Martin Avanzini (martin.avanzini@inria.fr)
Etienne Lozes (etienne.lozes@univ-cotedazur.fr)

MASTER INFORMATIQUE
UNIVERSITÉ CÔTE D'AZUR

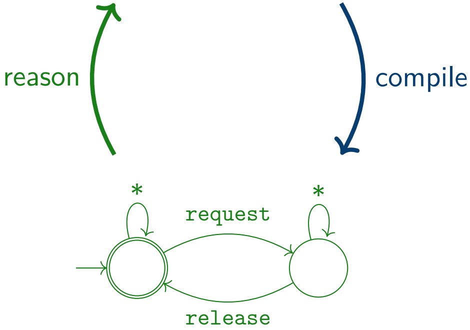
2nd Semester M1, 2022

Course Overview

- ⋆ based on the course given in 2019 by Etienne Lozes

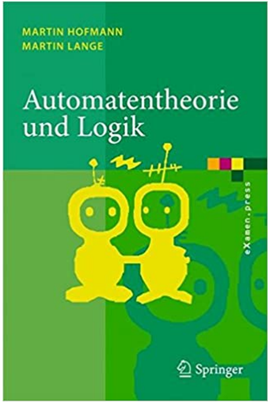


$$\forall x.\mathtt{request}(x) \rightarrow \exists y.x < y.\mathtt{release}(y)$$

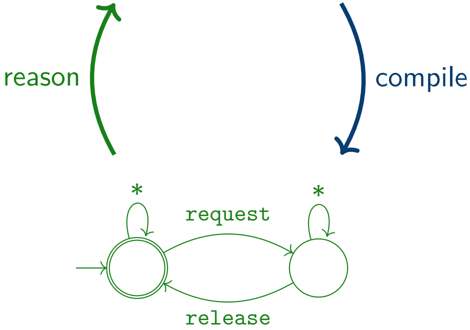


Course Overview

- ⋆ based on the course given in 2019 by Etienne Lozes



$$\forall x.\mathtt{request}(x) \rightarrow \exists y. x < y.\mathtt{release}(y)$$



- ⋆ no need to learn German, course material self-contained, available online

http://www-sop.inria.fr/members/Martin.Avanzini/teaching/2022/AL/

Course Overview

- ⋆ (non-)deterministic finite automata — Lecture 1
- ⋆ alternating finite automata — Lecture 2
- ⋆ (weak) monadic second order logic — Lectures 2,3

$$\exists X. 0 \in X \wedge \forall n. (n + 1 \in X \leftrightarrow n \notin X)$$

- ⋆ Presburger arithmetic — Lecture 3

$$\exists m. \exists n. m + n = 13 \wedge m = 1 + n$$

- ⋆ MONA tool — Lecture 4
- ⋆ Automata learning — Lecture 5
- ⋆ Büchi automata (infinite words) — Lecture 6
- ⋆ linear time logic — Lecture 7

$$\text{Globally}(\texttt{request} \rightarrow \text{Future}(\texttt{release}))$$

- ⋆ more stuff? just training? — Lecture 8

Administratives

1. 1/3 of lecture devoted to exercise 25%
   - approx. 2 hours of work between slots
   - solutions presented in class
   - participation in discussion counts towards final grade

2. two programming exercises 25%
   - you are free to pick your programming language
   - solutions presented in class

3. final exam 50%

Today's Lecture

Finite Word Automata Recap

1. regular languages and non-deterministic finite automata
2. closure properties, deterministic finite automata and Kleene's theorem
3. DFA equivalence and minimisation
4. decision procedures

Regular Languages and Non-Deterministic Finite Automata

Finite Words

- alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots\}$ is finite set of letters

- (finite) word $w = \mathtt{a}_1, \ldots, \mathtt{a}_n$ is finite sequence of letters $a_i \in \Sigma$
  - $|w| \triangleq n$ is length of word
  - $w[i] \triangleq \mathtt{a}_i$ denotes i-th letter in word w
  - ϵ is empty word of length 0
  - $v \cdot w$ (or simply vw) denotes concatenation of words v and w

$$\epsilon \cdot w = w = w \cdot \epsilon \qquad u \cdot (v \cdot w) = (u \cdot v) \cdot w$$

  - v^n is the word v concatenated with itself n times

Finite Words

- alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots\}$ is finite set of letters
- (finite) word $w = \mathtt{a}_1, \ldots, \mathtt{a}_\mathtt{n}$ is finite sequence of letters $a_i \in \Sigma$
  - $|w| \triangleq n$ is length of word
  - $w[i] \triangleq \mathtt{a}_\mathtt{i}$ denotes i-th letter in word w
  - ϵ is empty word of length 0
  - $v \cdot w$ (or simply vw) denotes concatenation of words v and w

$$\epsilon \cdot w = w = w \cdot \epsilon \qquad u \cdot (v \cdot w) = (u \cdot v) \cdot w$$

  - v^n is the word v concatenated with itself n times
- Σ^* denotes set of all words over alphabet Σ
- $\Sigma^+ \triangleq \Sigma^* \setminus \{\epsilon\}$ is set of non-empty words

Languages

- a language $L \subseteq \Sigma^*$ is a set of words
  - for instance, $\emptyset$, $\{\epsilon\}$, $\{\mathtt{aba}\}$, $\{\mathtt{a}, \mathtt{ab}, \mathtt{abb}, \mathtt{abbb}, \ldots\} = \{\mathtt{ab}^n \mid n \in \mathbb{N}\}$, Σ^* are all language

Languages

- a language $L \subseteq \Sigma^*$ is a set of words
  - for instance, $\varnothing$, $\{\epsilon\}$, $\{\mathtt{aba}\}$, $\{\mathtt{a}, \mathtt{ab}, \mathtt{abb}, \mathtt{abbb}, \dots\} = \{\mathtt{ab}^n \mid n \in \mathbb{N}\}$, Σ^* are all language
- new language definable from existing ones via set operations, e.g., if $L, M \subseteq \Sigma^*$:
  - union $L \cup M$, intersection $L \cap M$ and difference $L \setminus M$ are languages;

Languages

- ⋆ a language $L \subseteq \Sigma^*$ is a set of words
  - for instance, $\varnothing$, $\{\epsilon\}$, $\{\mathtt{aba}\}$, $\{\mathtt{a}, \mathtt{ab}, \mathtt{abb}, \mathtt{abbb}, \ldots\} = \{\mathtt{ab}^n \mid n \in \mathbb{N}\}$, Σ^* are all language
- ⋆ new language definable from existing ones via set operations, e.g., if $L, M \subseteq \Sigma^*$:
  - union $L \cup M$, intersection $L \cap M$ and difference $L \setminus M$ are languages;
  - complement $\overline{L} \triangleq \Sigma^* \setminus L$ forms a language

Languages

- a language $L \subseteq \Sigma^*$ is a set of words
  - for instance, $\varnothing$, $\{\epsilon\}$, $\{\mathtt{aba}\}$, $\{\mathtt{a}, \mathtt{ab}, \mathtt{abb}, \mathtt{abbb}, \ldots\} = \{\mathtt{ab}^n \mid n \in \mathbb{N}\}$, Σ^* are all language
- new language definable from existing ones via set operations, e.g., if $L, M \subseteq \Sigma^*$:
  - union $L \cup M$, intersection $L \cap M$ and difference $L \setminus M$ are languages;
  - complement $\overline{L} \triangleq \Sigma^* \setminus L$ forms a language
  - concatenation $L \cdot M$ yields a language, defined by concatenating all words in L with those in M:

$$L \cdot M \triangleq \{v \cdot w \mid v \in L \text{ and } w \in M\}$$

Languages

- a language $L \subseteq \Sigma^*$ is a set of words
  - for instance, $\varnothing$, $\{\epsilon\}$, $\{\mathtt{aba}\}$, $\{\mathtt{a}, \mathtt{ab}, \mathtt{abb}, \mathtt{abbb}, \dots\} = \{\mathtt{ab}^n \mid n \in \mathbb{N}\}$, Σ^* are all language
- new language definable from existing ones via set operations, e.g., if $L, M \subseteq \Sigma^*$:
  - union $L \cup M$, intersection $L \cap M$ and difference $L \setminus M$ are languages;
  - complement $\overline{L} \triangleq \Sigma^* \setminus L$ forms a language
  - concatenation $L \cdot M$ yields a language, defined by concatenating all words in L with those in M:

$$L \cdot M \triangleq \{v \cdot w \mid v \in L \text{ and } w \in M\}$$

  - Kleene Star L^* yields a language, defined as

$$L^* \triangleq \bigcup_{n \in \mathbb{N}} L^n \quad \text{where } L^0 \triangleq \{\epsilon\} \text{ and } L^{n+1} = L \cdot L^n$$

    for instance

$$\{\mathtt{ab}, \mathtt{c}\}^* = \{\epsilon, \quad \mathtt{ab}, \mathtt{c}, \quad \mathtt{abab}, \mathtt{abc}, \mathtt{cab}, \mathtt{cc}, quad \dots\}$$

Regular Languages

The class $REG(\Sigma)$ of regular languages is the *smallest* class (i.e., set of) languages s.t.

1. $\varnothing \in REG(\Sigma)$ and $\{\mathtt{a}\} \in REG(\Sigma)$ for every $\mathtt{a} \in \Sigma$; and
2. if $L, M \in REG(\Sigma)$ then $L \cup M \in REG(\Sigma)$, $L \cdot M \in REG(\Sigma)$ and $L^* \in REG(\Sigma)$.

Regular Languages

The class $REG(\Sigma)$ of regular languages is the *smallest* class (i.e., set of) languages s.t.

1. $\varnothing \in REG(\Sigma)$ and $\{\mathtt{a}\} \in REG(\Sigma)$ for every $\mathtt{a} \in \Sigma$; and
2. if $L, M \in REG(\Sigma)$ then $L \cup M \in REG(\Sigma)$, $L \cdot M \in REG(\Sigma)$ and $L^* \in REG(\Sigma)$.

Examples

- ⋆ $\{\epsilon\} = \varnothing^*$ is regular
- ⋆ $\{\epsilon\} \cup ((\{\mathtt{a}\} \cup \{\mathtt{b}\})^* \cdot \{\mathtt{b}\})$, or $\epsilon \cup (\mathtt{a} \cup \mathtt{b})^*\mathtt{b}$ for short, is regular
- ⋆ every finite language $L = \{w_1, \ldots, w_n\}$ is regular

Regular Languages

The class $REG(\Sigma)$ of regular languages is the *smallest* class (i.e., set of) languages s.t.

1. $\varnothing \in REG(\Sigma)$ and $\{\mathtt{a}\} \in REG(\Sigma)$ for every $\mathtt{a} \in \Sigma$; and
2. if $L, M \in REG(\Sigma)$ then $L \cup M \in REG(\Sigma)$, $L \cdot M \in REG(\Sigma)$ and $L^* \in REG(\Sigma)$.

Examples

- $\{\epsilon\} = \varnothing^*$ is regular
- $\{\epsilon\} \cup ((\{\mathtt{a}\} \cup \{\mathtt{b}\})^* \cdot \{\mathtt{b}\})$, or $\epsilon \cup (\mathtt{a} \cup \mathtt{b})^*\mathtt{b}$ for short, is regular
- every finite language $L = \{w_1, \ldots, w_n\}$ is regular

Note

- apart from those named in (2), $REG(\Sigma)$ is closed under many more operations (particularly: intersection, complement)
- to show such closure properties, it is convenient to have a suitable characterisation

Non-deterministic Finite Automata

A non-deterministic finite automata (NFA) $\mathcal{A}$ is a tuple $(Q, \Sigma, q_I, \delta, F)$ consisting of

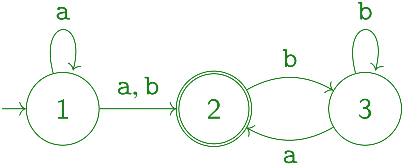
- ⋆ a finite set of states Q
- ⋆ an alphabet Σ
- ⋆ an initial state $q_I \in Q$
- ⋆ a transition function $\delta : Q \times \Sigma \to 2^Q$
- ⋆ a set of final states $F \subseteq Q$

Non-deterministic Finite Automata

A non-deterministic finite automata (NFA) $\mathcal{A}$ is a tuple $(Q, \Sigma, q_I, \delta, F)$ consisting of

- ⋆ a finite set of states Q
- ⋆ an alphabet Σ
- ⋆ an initial state $q_I \in Q$
- ⋆ a transition function $\delta : Q \times \Sigma \to 2^Q$
- ⋆ a set of final states $F \subseteq Q$

Represented often as graph:



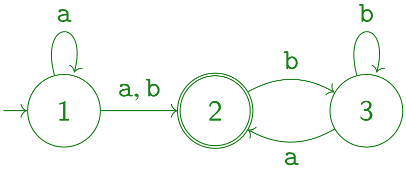
δ	a	b
1	$\{1,2\}$	$\{2\}$
2	$\varnothing$	$\{3\}$
3	$\{2\}$	$\{3\}$

Non-deterministic Finite Automata

A non-deterministic finite automata (NFA) $\mathcal{A}$ is a tuple $(Q, \Sigma, q_I, \delta, F)$ consisting of

- a finite set of states Q
- an alphabet Σ
- an initial state $q_I \in Q$
- a transition function $\delta : Q \times \Sigma \to 2^Q$
- a set of final states $F \subseteq Q$

Represented often as graph:



δ	a	b
1	$\{1,2\}$	$\{2\}$
2	$\varnothing$	$\{3\}$
3	$\{2\}$	$\{3\}$

Notation: $p \xrightarrow{\mathtt{a}} q$ if $q \in \delta(p, \mathtt{a})$

Language Recognized by NFA

Consider NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$

- ⋆ if q_0 is initial state q_I then $q_I = q_0 \xrightarrow{\mathtt{a}_1} q_1 \xrightarrow{\mathtt{a}_2} \cdots \xrightarrow{\mathtt{a}_n} q_n$ is called run on $w = \mathtt{a}_1 \ldots \mathtt{a}_n$

Language Recognized by NFA

Consider NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$

- ⋆ if q_0 is initial state q_I then $q_I = q_0 \xrightarrow{\mathtt{a}_1} q_1 \xrightarrow{\mathtt{a}_2} \cdots \xrightarrow{\mathtt{a}_n} q_n$ is called run on $w = \mathtt{a}_1 \ldots \mathtt{a}_n$
- ⋆ run is accepting if $q_n \in F$ is final

Language Recognized by NFA

Consider NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$

- if q_0 is initial state q_I then $q_I = q_0 \xrightarrow{\mathtt{a}_1} q_1 \xrightarrow{\mathtt{a}_2} \cdots \xrightarrow{\mathtt{a}_n} q_n$ is called run on $w = \mathtt{a}_1 \ldots \mathtt{a}_n$
- run is accepting if $q_n \in F$ is final
- language $L(\mathcal{A})$ recognized by $\mathcal{A}$ consists of all words that have accepting run

$$L(\mathcal{A}) \triangleq \{w \mid \delta^*(q_I, w) \cap F \neq \emptyset\}$$

where extended transition function $\delta^* : Q \times \Sigma^* \to 2^Q$ defined such that

$$q \in \delta^*(p, \mathtt{a}_1 \ldots \mathtt{a}_n) \text{ iff } p = q_0 \xrightarrow{\mathtt{a}_1} q_1 \xrightarrow{\mathtt{a}_2} \ldots \xrightarrow{\mathtt{a}_n} q_n = q$$

Language Recognized by NFA

Consider NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$

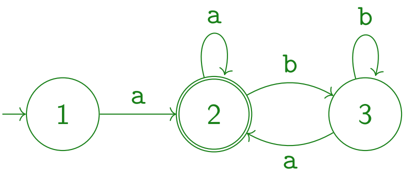
- ⋆ if q_0 is initial state q_I then $q_I = q_0 \xrightarrow{\mathtt{a}_1} q_1 \xrightarrow{\mathtt{a}_2} \cdots \xrightarrow{\mathtt{a}_n} q_n$ is called run on $w = \mathtt{a}_1 \ldots \mathtt{a}_n$
- ⋆ run is accepting if $q_n \in F$ is final
- ⋆ language $\mathrm{L}(\mathcal{A})$ recognized by $\mathcal{A}$ consists of all words that have accepting run

$$\mathrm{L}(\mathcal{A}) \triangleq \{w \mid \delta^*(q_I, w) \cap F \neq \varnothing\}$$

where extended transition function $\delta^* : Q \times \Sigma^* \to 2^Q$ defined such that

$$q \in \delta^*(p, \mathtt{a}_1 \ldots \mathtt{a}_n) \text{ iff } p = q_0 \xrightarrow{\mathtt{a}_1} q_1 \xrightarrow{\mathtt{a}_2} \ldots \xrightarrow{\mathtt{a}_n} q_n = q$$

Example



$\mathrm{L}(\mathcal{A}) = ?$

Language Recognized by NFA

Consider NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$

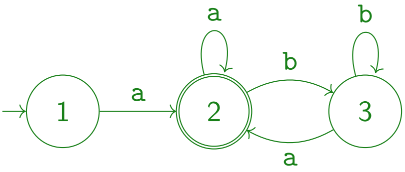
- ⋆ if q_0 is initial state q_I then $q_I = q_0 \xrightarrow{\mathtt{a}_1} q_1 \xrightarrow{\mathtt{a}_2} \cdots \xrightarrow{\mathtt{a}_n} q_n$ is called run on $w = \mathtt{a}_1 \dots \mathtt{a}_n$
- ⋆ run is accepting if $q_n \in F$ is final
- ⋆ language $\mathrm{L}(\mathcal{A})$ recognized by $\mathcal{A}$ consists of all words that have accepting run

$$\mathrm{L}(\mathcal{A}) \triangleq \{w \mid \delta^*(q_I, w) \cap F \neq \varnothing\}$$

where extended transition function $\delta^* : Q \times \Sigma^* \to 2^Q$ defined such that

$$q \in \delta^*(p, \mathtt{a}_1 \dots \mathtt{a}_n) \text{ iff } p = q_0 \xrightarrow{\mathtt{a}_1} q_1 \xrightarrow{\mathtt{a}_2} \dots \xrightarrow{\mathtt{a}_n} q_n = q$$

Example



$\mathrm{L}(\mathcal{A}) = ?$

Language Recognized by NFA

Consider NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$

- if q_0 is initial state q_I then $q_I = q_0 \xrightarrow{\mathtt{a}_1} q_1 \xrightarrow{\mathtt{a}_2} \cdots \xrightarrow{\mathtt{a}_n} q_n$ is called run on $w = \mathtt{a}_1 \ldots \mathtt{a}_n$
- run is accepting if $q_n \in F$ is final

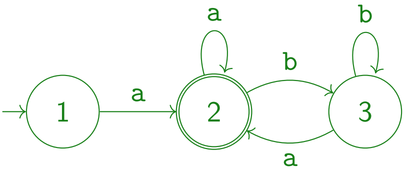
- language $\mathrm{L}(\mathcal{A})$ recognized by $\mathcal{A}$ consists of all words that have accepting run

$$\mathrm{L}(\mathcal{A}) \triangleq \{ w \mid \delta^*(q_I, w) \cap F \neq \varnothing \}$$

where extended transition function $\delta^* : Q \times \Sigma^* \to 2^Q$ defined such that

$$q \in \delta^*(p, \mathtt{a}_1 \ldots \mathtt{a}_n) \text{ iff } p = q_0 \xrightarrow{\mathtt{a}_1} q_1 \xrightarrow{\mathtt{a}_2} \ldots \xrightarrow{\mathtt{a}_n} q_n = q$$

Example



$\mathrm{L}(\mathcal{A}) = \{ w \in \Sigma^+ \mid w \text{ starts and ends with } \mathtt{a} \}$

Closure Properties, Deterministic Finite Automata and Kleene's Theorem

Closure Properties

A language L is recognizable if there is an NFA $\mathcal{A}$ with $\mathrm{L}(\mathcal{A}) = L$

Theorem (Closure Properties of NFAs)

For recognizable L, M, the following are recognizable:

1. *union* $L \cup M$
2. *concatenation* $L \cdot M$
3. *Kleene's star* L^*
4. *intersection* $L \cap M$
5. *complement* $\overline{L}$

Closure Properties

A language L is recognizable if there is an NFA $\mathcal{A}$ with $\mathrm{L}(\mathcal{A}) = L$

Theorem (Closure Properties of NFAs)

For recognizable L, M, the following are recognizable:

1. *union* $L \cup M$
2. *concatenation* $L \cdot M$
3. *Kleene's star* L^*
4. *intersection* $L \cap M$
5. *complement* $\overline{L}$

Proof Outline.

- ⋆ (1)–(4) follow from a construction (see exercise, next slide)
- ⋆ (5) translate to deterministic automaton (why can't we simply invert final states?)

Closure Properties

A language L is recognizable if there is an NFA $\mathcal{A}$ with $\mathrm{L}(\mathcal{A}) = L$

Theorem (Closure Properties of NFAs)

For recognizable L, M, the following are recognizable:

1. *union* $L \cup M$
2. *concatenation* $L \cdot M$
3. *Kleene's star* L^*
4. *intersection* $L \cap M$
5. *complement* $\overline{L}$

Proof Outline.

- ⋆ (1)–(4) follow from a construction (see exercise, next slide)
- ⋆ (5) translate to deterministic automaton (why can't we simply invert final states?)

Note

- ⋆ the class of recognized languages forms a Boolean Algebra

Closure Properties
Kleene's Star

Lemma

If L is recognizable, then so is L^*.

Proof Outline.

For NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$ recognizing L, define $\mathcal{A}^* \triangleq (Q \uplus \{q'\}, \Sigma, q', \delta', F \cup \{q'\})$ where

$$\delta'(q', a) \triangleq \delta(q_I, a) \qquad \delta'(q, a) \triangleq \begin{cases} \delta(q, a) \cup \delta(q_I, a) & \text{if } q \in F; \\ \delta(q, a) & \text{if } q \in Q \setminus F. \end{cases}$$

Characterisation of *REG*

Theorem

NFAs recognize precisely the regular languages $REG(\Sigma)$.

Characterisation of *REG*

Theorem

NFAs recognize precisely the regular languages $REG(\Sigma)$.

Proof Outline.

$\Leftarrow$ By induction on $REG(\Sigma)$, using closure properties. (how, why?)

Characterisation of *REG*

Theorem

NFAs recognize precisely the regular languages $REG(\Sigma)$.

Proof Outline.

$\Leftarrow$ By induction on $REG(\Sigma)$, using closure properties. (how, why?)

$\Rightarrow$ Fix NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$.

- For $p \in Q$, start with equations

$$L(p) = \bigcup_{p \xrightarrow{\mathtt{a}} q} \mathtt{a} \cdot L(q) \cup \begin{cases} \{\epsilon\} & \text{if } p \text{ final;} \\ \varnothing & \text{otherwise.} \end{cases}$$

- (intuition?)

Characterisation of *REG*

Theorem

NFAs recognize precisely the regular languages $REG(\Sigma)$.

Proof Outline.

$\Leftarrow$ By induction on $REG(\Sigma)$, using closure properties. (how, why?)

$\Rightarrow$ Fix NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$.

- For $p \in Q$, start with equations

$$L(p) = \bigcup_{p \xrightarrow{\mathrm{a}} q} \mathrm{a} \cdot L(q) \cup \begin{cases} \{\epsilon\} & \text{if } p \text{ final;} \\ \varnothing & \text{otherwise.} \end{cases}$$

- thus $L(p)$ collects words $w = \mathrm{a}_1 \dots \mathrm{a}_\mathrm{n}$ s.t. $p = q_0 \xrightarrow{\mathrm{a}_1} q_1 \xrightarrow{\mathrm{a}_2} \cdots \xrightarrow{\mathrm{a}_\mathrm{n}} q_n \in F$

Characterisation of REG

Theorem

NFAs recognize precisely the regular languages $REG(\Sigma)$.

Proof Outline.

$\Leftarrow$ By induction on $REG(\Sigma)$, using closure properties. (how, why?)

$\Rightarrow$ Fix NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$.

- For $p \in Q$, start with equations

$$L(p) = \bigcup_{p \xrightarrow{\mathtt{a}} q} \mathtt{a} \cdot L(q) \cup \begin{cases} \{\epsilon\} & \text{if } p \text{ final;} \\ \varnothing & \text{otherwise.} \end{cases}$$

- thus $L(p)$ collects words $w = \mathtt{a}_1 \ldots \mathtt{a}_n$ s.t. $p = q_0 \xrightarrow{\mathtt{a}_1} q_1 \xrightarrow{\mathtt{a}_2} \cdots \xrightarrow{\mathtt{a}_n} q_n \in F$
- pick $p \in Q$ and apply Arden's Equality

$$L(p) = M \cdot L(p) \cup N \quad \Longrightarrow \quad L(p) = M^* \cdot N \tag{1}$$

Characterisation of REG

Theorem

NFAs recognize precisely the regular languages $REG(\Sigma)$.

Proof Outline.

$\Leftarrow$ By induction on $REG(\Sigma)$, using closure properties. (how, why?)

$\Rightarrow$ Fix NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$.

- For $p \in Q$, start with equations

$$L(p) = \bigcup_{p \xrightarrow{\mathtt{a}} q} \mathtt{a} \cdot L(q) \cup \begin{cases} \{\epsilon\} & \text{if } p \text{ final;} \\ \varnothing & \text{otherwise.} \end{cases}$$

- thus $L(p)$ collects words $w = \mathtt{a}_1 \ldots \mathtt{a}_\mathtt{n}$ s.t. $p = q_0 \xrightarrow{\mathtt{a}_1} q_1 \xrightarrow{\mathtt{a}_2} \cdots \xrightarrow{\mathtt{a}_\mathtt{n}} q_n \in F$
- pick $p \in Q$ and apply Arden's Equality

$$L(p) = M \cdot L(p) \cup N \quad \Rightarrow \quad L(p) = M^* \cdot N \tag{1}$$

- simplify; substitute and repeat until (1) not applicable

Characterisation of *REG*

Theorem

NFAs recognize precisely the regular languages $REG(\Sigma)$.

Proof Outline.

$\Leftarrow$ By induction on $REG(\Sigma)$, using closure properties. (how, why?)

$\Rightarrow$ Fix NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$.

- For $p \in Q$, start with equations

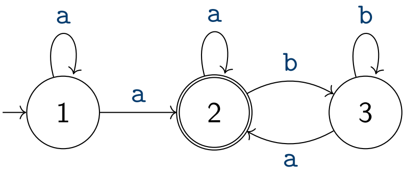
$$L(p) = \bigcup_{p \xrightarrow{\mathrm{a}} q} \mathrm{a} \cdot L(q) \cup \begin{cases} \{\epsilon\} & \text{if } p \text{ final;} \\ \varnothing & \text{otherwise.} \end{cases}$$

- thus $L(p)$ collects words $w = \mathrm{a}_1 \ldots \mathrm{a}_\mathrm{n}$ s.t. $p = q_0 \xrightarrow{\mathrm{a}_1} q_1 \xrightarrow{\mathrm{a}_2} \cdots \xrightarrow{\mathrm{a}_\mathrm{n}} q_n \in F$
- pick $p \in Q$ and apply Arden's Equality

$$L(p) = M \cdot L(p) \cup N \quad \Rightarrow \quad L(p) = M^* \cdot N \tag{1}$$

- simplify; substitute and repeat until (1) not applicable
- $L(q_I) = \mathrm{L}(\mathcal{A})$ eventually in $REG(\Sigma)$

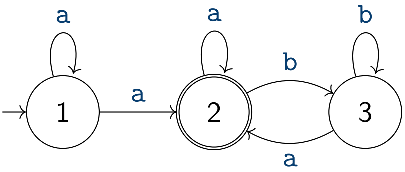
Example



$$L(1) = \mathtt{a}L(1) \cup \mathtt{a}L(2) \qquad L(2) = \mathtt{a}L(2) \cup \mathtt{b}L(3) \cup \epsilon \qquad L(3) = \mathtt{a}L(2) \cup \mathtt{b}L(3)$$

$$\Longrightarrow \quad L(1) = \mathtt{a}^*\mathtt{a}L(2)$$

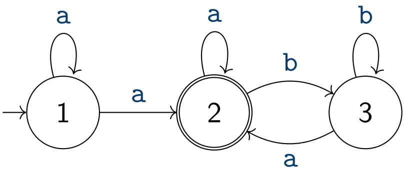
Example



$$L(1) = \mathtt{a}L(1) \cup \mathtt{a}L(2) \qquad L(2) = \mathtt{a}L(2) \cup \mathtt{b}L(3) \cup \epsilon \qquad L(3) = \mathtt{a}L(2) \cup \mathtt{b}L(3)$$

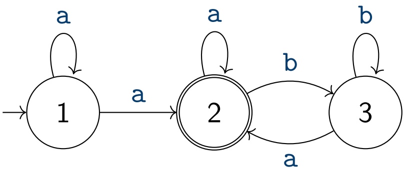
$$\Longrightarrow \quad L(1) = \mathtt{a}^*\mathtt{a}L(2) \qquad L(2) = \mathtt{a}^*(\mathtt{b}L(3) \cup \epsilon)$$

Example



$$
\begin{array}{llll}
 & L(1) = \mathtt{a}L(1) \cup \mathtt{a}L(2) & L(2) = \mathtt{a}L(2) \cup \mathtt{b}L(3) \cup \epsilon & L(3) = \mathtt{a}L(2) \cup \mathtt{b}L(3) \\
\Longrightarrow & L(1) = \mathtt{a}^*\mathtt{a}L(2) & L(2) = \mathtt{a}^*(\mathtt{b}L(3) \cup \epsilon) & L(3) = \mathtt{b}^*\mathtt{a}L(2) \\
\Longrightarrow & & &
\end{array}
$$

Example



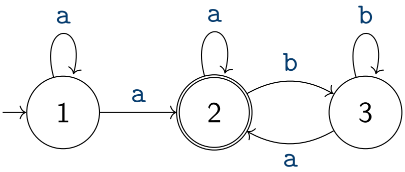
$$L(1) = \mathtt{a}L(1) \cup \mathtt{a}L(2) \qquad L(2) = \mathtt{a}L(2) \cup \mathtt{b}L(3) \cup \epsilon \qquad L(3) = \mathtt{a}L(2) \cup \mathtt{b}L(3)$$

$$\Longrightarrow \quad L(1) = \mathtt{a}^*\mathtt{a}L(2) \qquad L(2) = \mathtt{a}^*(\mathtt{b}L(3) \cup \epsilon) \qquad L(3) = \mathtt{b}^*\mathtt{a}L(2)$$

$$\Longrightarrow \quad L(1) = \mathtt{a}^*\mathtt{a}L(2) \qquad L(2) = \mathtt{a}^*(\mathtt{b}\mathtt{b}^*\mathtt{a}L(2) \cup \epsilon)$$

$$\Longrightarrow$$

Example



$$L(1) = \mathtt{a}L(1) \cup \mathtt{a}L(2) \qquad L(2) = \mathtt{a}L(2) \cup \mathtt{b}L(3) \cup \epsilon \qquad L(3) = \mathtt{a}L(2) \cup \mathtt{b}L(3)$$

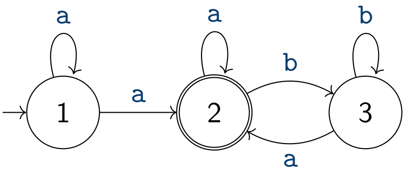
$$\Longrightarrow \quad L(1) = \mathtt{a}^*\mathtt{a}L(2) \qquad L(2) = \mathtt{a}^*(\mathtt{b}L(3) \cup \epsilon) \qquad L(3) = \mathtt{b}^*\mathtt{a}L(2)$$

$$\Longrightarrow \quad L(1) = \mathtt{a}^*\mathtt{a}L(2) \qquad L(2) = \mathtt{a}^*(\mathtt{bb}^*\mathtt{a}L(2) \cup \epsilon)$$

$$\Longrightarrow \quad L(1) = \mathtt{a}^*\mathtt{a}L(2) \qquad L(2) = \mathtt{a}^*\mathtt{bb}^*\mathtt{a}L(2) \cup \mathtt{a}^*$$

$$\Longrightarrow$$

Example



$$L(1) = aL(1) \cup aL(2) \qquad L(2) = aL(2) \cup bL(3) \cup \epsilon \qquad L(3) = aL(2) \cup bL(3)$$

$$\Rightarrow \quad L(1) = a^*aL(2) \qquad L(2) = a^*(bL(3) \cup \epsilon) \qquad L(3) = b^*aL(2)$$

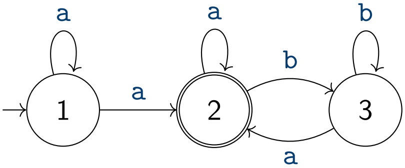
$$\Rightarrow \quad L(1) = a^*aL(2) \qquad L(2) = a^*(bb^*aL(2) \cup \epsilon)$$

$$\Rightarrow \quad L(1) = a^*aL(2) \qquad L(2) = a^*bb^*aL(2) \cup a^*$$

$$\Rightarrow \quad L(1) = a^*aL(2) \qquad L(2) = (a^*bb^*a)^*a^*$$

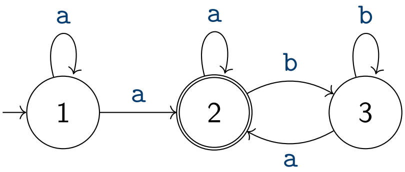
$$\Rightarrow$$

Example



$$
\begin{array}{rlll}
 & L(1) = \mathtt{a}L(1) \cup \mathtt{a}L(2) & L(2) = \mathtt{a}L(2) \cup \mathtt{b}L(3) \cup \epsilon & L(3) = \mathtt{a}L(2) \cup \mathtt{b}L(3) \\
\Longrightarrow & L(1) = \mathtt{a}^*\mathtt{a}L(2) & L(2) = \mathtt{a}^*(\mathtt{b}L(3) \cup \epsilon) & L(3) = \mathtt{b}^*\mathtt{a}L(2) \\
\Longrightarrow & L(1) = \mathtt{a}^*\mathtt{a}L(2) & L(2) = \mathtt{a}^*(\mathtt{b}\mathtt{b}^*\mathtt{a}L(2) \cup \epsilon) & \\
\Longrightarrow & L(1) = \mathtt{a}^*\mathtt{a}L(2) & L(2) = \mathtt{a}^*\mathtt{b}\mathtt{b}^*\mathtt{a}L(2) \cup \mathtt{a}^* & \\
\Longrightarrow & L(1) = \mathtt{a}^*\mathtt{a}L(2) & L(2) = (\mathtt{a}^*\mathtt{b}\mathtt{b}^*\mathtt{a})^*\mathtt{a}^* & \\
\Longrightarrow & L(1) = \mathtt{a}^*\mathtt{a}(\mathtt{a}^*\mathtt{b}\mathtt{b}^*\mathtt{a})^*\mathtt{a}^* & & \\
\Longrightarrow & & &
\end{array}
$$

Example



$$
\begin{array}{lll}
& L(1) = \mathtt{a}L(1) \cup \mathtt{a}L(2) & L(2) = \mathtt{a}L(2) \cup \mathtt{b}L(3) \cup \epsilon & L(3) = \mathtt{a}L(2) \cup \mathtt{b}L(3) \\
\Longrightarrow & L(1) = \mathtt{a}^*\mathtt{a}L(2) & L(2) = \mathtt{a}^*(\mathtt{b}L(3) \cup \epsilon) & L(3) = \mathtt{b}^*\mathtt{a}L(2) \\
\Longrightarrow & L(1) = \mathtt{a}^*\mathtt{a}L(2) & L(2) = \mathtt{a}^*(\mathtt{b}\mathtt{b}^*\mathtt{a}L(2) \cup \epsilon) & \\
\Longrightarrow & L(1) = \mathtt{a}^*\mathtt{a}L(2) & L(2) = \mathtt{a}^*\mathtt{b}\mathtt{b}^*\mathtt{a}L(2) \cup \mathtt{a}^* & \\
\Longrightarrow & L(1) = \mathtt{a}^*\mathtt{a}L(2) & L(2) = (\mathtt{a}^*\mathtt{b}\mathtt{b}^*\mathtt{a})^*\mathtt{a}^* & \\
\Longrightarrow & L(1) = \mathtt{a}^*\mathtt{a}(\mathtt{a}^*\mathtt{b}\mathtt{b}^*\mathtt{a})^*\mathtt{a}^* & & \\
\Longrightarrow & L(1) = \mathtt{a}\mathtt{a}^*(\mathtt{b}\mathtt{b}^*\mathtt{a}\mathtt{a}^*)^*\mathtt{a}^* & & \\
\Longrightarrow & L(1) = \mathtt{a}^+(\mathtt{b}^+\mathtt{a}^+)^* & &
\end{array}
$$

Deterministic Finite Automata

A deterministic finite automata (DFA) $\mathcal{A}$ is a NFA where each state has precisely one successor state:

$$\delta : Q \times \Sigma \rightarrow Q$$

Deterministic Finite Automata

A deterministic finite automata (DFA) $\mathcal{A}$ is a NFA where each state has precisely one successor state:

$$\delta : Q \times \Sigma \to Q$$

Theorem (Determinisation)

A language is recognizable by an NFA if and only if it is recognizable by a DFA.

Deterministic Finite Automata

A deterministic finite automata (DFA) $\mathcal{A}$ is a NFA where each state has precisely one successor state:

$$\delta : Q \times \Sigma \to Q$$

Theorem (Determinisation)

A language is recognizable by an NFA if and only if it is recognizable by a DFA.

Proof Outline.

$\Leftarrow$ Every DFA is an NFA.

$\Rightarrow$ Given NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$ recognizing L, define DFA $\mathcal{A}_d(2^Q, \Sigma, \{q_I\}, \delta_d, F_d)$ s.t.:

- $\delta_d(\{q_1, \ldots, q_n\}, \mathrm{a}) \triangleq \delta(q_1, \mathrm{a}) \cup \cdots \cup \delta(q_n, \mathrm{a})$
- $F_d \triangleq \{S \subseteq Q \mid F \cap S \neq \varnothing\}$, i.e., $\{q_1, \ldots, q_n\}$ final in $\mathcal{A}_d$ if one of the q_i final in $\mathcal{A}$

Then $\mathcal{A}_d$ recognizes L:

$$\text{run in new } \mathcal{A}_d \text{ on word } w \quad \equiv \quad \textit{all} \text{ runs on } w \text{ in } \mathcal{A}$$

Deterministic Finite Automata

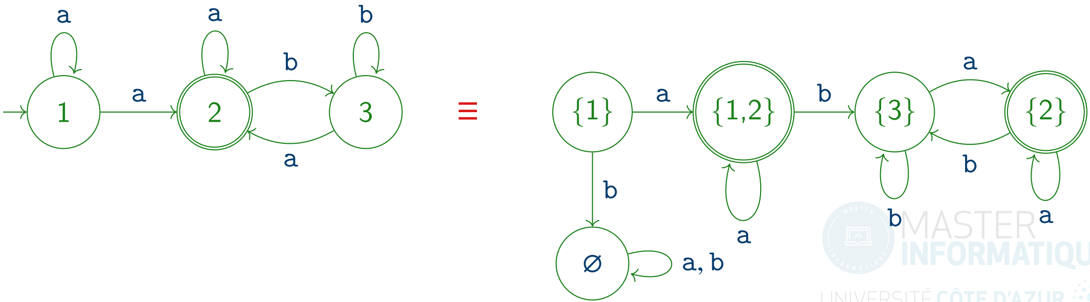
A deterministic finite automata (DFA) $\mathcal{A}$ is a NFA where each state has precisely one successor state:

$$\delta : Q \times \Sigma \to Q$$

Theorem (Determinisation)

A language is recognizable by an NFA if and only if it is recognizable by a DFA.

Example



Deterministic Finite Automata

A deterministic finite automata (DFA) $\mathcal{A}$ is a NFA where each state has precisely one successor state:

$$\delta : Q \times \Sigma \to Q$$

Theorem (Determinisation)

A language is recognizable by an NFA if and only if it is recognizable by a DFA.

Lemma

If L is regular, then so its complement $\overline{L} = \Sigma^* \setminus L$.

Proof Outline.

ideas?

Deterministic Finite Automata

A deterministic finite automata (DFA) $\mathcal{A}$ is a NFA where each state has precisely one successor state:

$$\delta : Q \times \Sigma \to Q$$

Theorem (Determinisation)

A language is recognizable by an NFA if and only if it is recognizable by a DFA.

Lemma

If L is regular, then so its complement $\overline{L} = \Sigma^* \setminus L$.

Proof Outline.

- Since L is regular, there is a DFA $\mathcal{A}$ with $\mathrm{L}(\mathcal{A}) = L$
- flipping the set of final states in $\mathcal{A}$ results in DFA $\overline{\mathcal{A}}$ with $\mathrm{L}(\overline{\mathcal{A}}) = \overline{L}$

Kleene's Theorem

Theorem

The following are equivalent:

1. *The class of regular languages* $REG(\Sigma)$
2. *The class of languages recognized by NFAs over* Σ
3. *The class of languages recognized by DFAs over* Σ

An Unpleasant Theorem

Theorem

For every number $n \in \mathbb{N}$ there exists an NFA $\mathcal{A}$ with $n + 1$ states such that every equivalent DFA has at least 2^n states.

$\Longrightarrow$ NFAs can be exponentially more succinct than DFAs

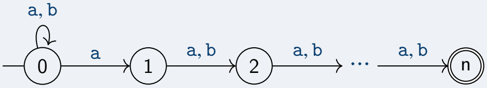
An Unpleasant Theorem

Theorem

For every number $n \in \mathbb{N}$ there exists an NFA $\mathcal{A}$ with $n + 1$ states such that every equivalent DFA has at least 2^n states.

Proof Outline.

- ★ consider the NFA



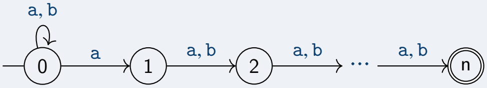
An Unpleasant Theorem

Theorem

For every number $n \in \mathbb{N}$ there exists an NFA $\mathcal{A}$ with $n+1$ states such that every equivalent DFA has at least 2^n states.

Proof Outline.

- consider the NFA



- for a proof by contradiction, suppose equivalent DFA $\mathcal{A}$ has strictly less than 2^n states:

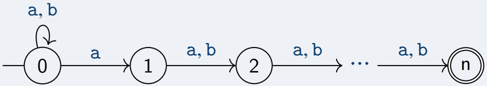
An Unpleasant Theorem

Theorem

For every number $n \in \mathbb{N}$ there exists an NFA $\mathcal{A}$ with $n+1$ states such that every equivalent DFA has at least 2^n states.

Proof Outline.

- ⋆ consider the NFA



- ⋆ for a proof by contradiction, suppose equivalent DFA $\mathcal{A}$ has strictly less than 2^n states:
  - since there are 2^n words of length n, there must be two such distinct words $u, v \in \Sigma^n$ ending up in the same state, i.e. $\delta^*(q_I, u) = \delta^*(q_I, v)$

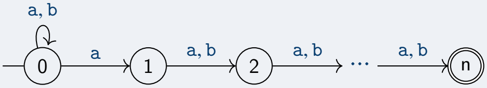
An Unpleasant Theorem

Theorem

For every number $n \in \mathbb{N}$ there exists an NFA $\mathcal{A}$ with $n+1$ states such that every equivalent DFA has at least 2^n states.

Proof Outline.

- consider the NFA



- for a proof by contradiction, suppose equivalent DFA $\mathcal{A}$ has strictly less than 2^n states:
  - since there are 2^n words of length n, there must be two such distinct words $u, v \in \Sigma^n$ ending up in the same state, i.e. $\delta^*(q_I, u) = \delta^*(q_I, v)$
  - suppose they differ at position i, e.g., $u[i] = \mathtt{a}$ and $v[i] = \mathtt{b}$, hence

$$u \underbrace{\mathtt{a}\cdots\mathtt{a}}_{i-1 \text{ times}} \in L(\mathcal{A}) \qquad \text{but} \qquad v \underbrace{\mathtt{a}\cdots\mathtt{a}}_{i-1 \text{ times}} \notin L(\mathcal{A})$$

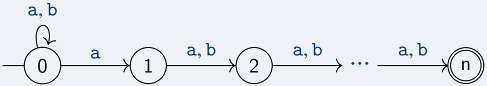
An Unpleasant Theorem

Theorem

For every number $n \in \mathbb{N}$ there exists an NFA $\mathcal{A}$ with $n+1$ states such that every equivalent DFA has at least 2^n states.

Proof Outline.

- consider the NFA



- for a proof by contradiction, suppose equivalent DFA $\mathcal{A}$ has strictly less than 2^n states:
  - since there are 2^n words of length n, there must be two such distinct words $u, v \in \Sigma^n$ ending up in the same state, i.e. $\delta^*(q_I, u) = \delta^*(q_I, v)$
  - suppose they differ at position i, e.g., $u[i] = \mathtt{a}$ and $v[i] = \mathtt{b}$, hence

$$u \underbrace{\mathtt{a} \cdots \mathtt{a}}_{i-1 \text{ times}} \in \mathrm{L}(\mathcal{A}) \qquad \text{but} \qquad v \underbrace{\mathtt{a} \cdots \mathtt{a}}_{i-1 \text{ times}} \notin \mathrm{L}(\mathcal{A})$$

  - the DFA now either accepts or rejects both extended words; contradicting that $\mathcal{A}$ is equivalent to the NFA

DFA Equivalence and Minimisation

DFA Equivalence and Minimisation

Two NFAs (DFAs) $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent if $L(\mathcal{A}_1) = L(\mathcal{A}_2)$.

DFA Equivalence and Minimisation

Two NFAs (DFAs) $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent if $L(\mathcal{A}_1) = L(\mathcal{A}_2)$.

Theorem

For every DFA there exists a unique (up to renaming of states) minimal DFA.

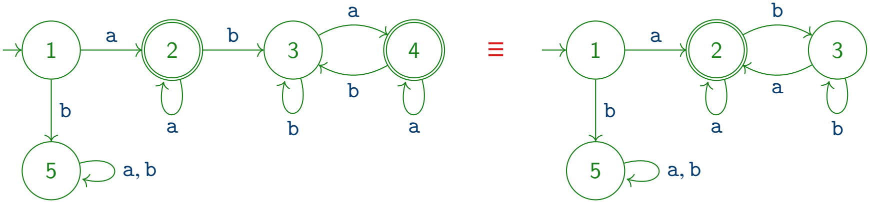
DFA Equivalence and Minimisation

Two NFAs (DFAs) $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent if $L(\mathcal{A}_1) = L(\mathcal{A}_2)$.

Theorem

For every DFA there exists a unique (up to renaming of states) minimal DFA.

Example



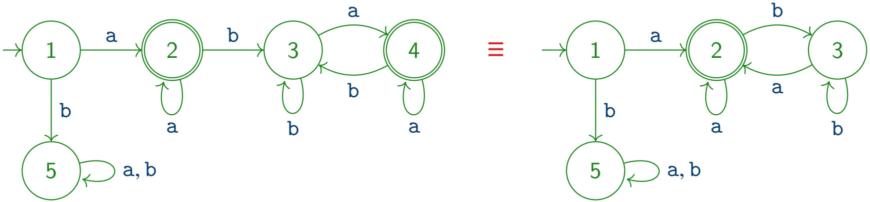
DFA Equivalence and Minimisation

Two NFAs (DFAs) $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent if $\mathrm{L}(\mathcal{A}_1) = \mathrm{L}(\mathcal{A}_2)$.

Theorem

For every DFA there exists a unique (up to renaming of states) minimal DFA.

Example



- let $\mathrm{L}(p, \mathcal{A}) \triangleq \{w \mid \delta^*(p, w) \in F\}$, hence in particular, $\mathrm{L}(\mathcal{A}) = \mathrm{L}(q_I, \mathcal{A})$
- two states p, q are equivalent in $\mathcal{A}$ if accepting runs coincide:

$$p \equiv_{\mathcal{A}} q \qquad :\Longleftrightarrow \qquad \mathrm{L}(p, \mathcal{A}) = \mathrm{L}(q, \mathcal{A})$$

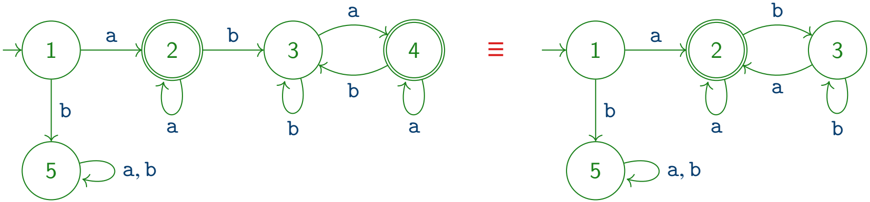
DFA Equivalence and Minimisation

Two NFAs (DFAs) $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent if $\mathrm{L}(\mathcal{A}_1) = \mathrm{L}(\mathcal{A}_2)$.

Theorem

For every DFA there exists a unique (up to renaming of states) minimal DFA.

Example



- ⋆ let $\mathrm{L}(p, \mathcal{A}) \triangleq \{w \mid \delta^*(p, w) \in F\}$, hence in particular, $\mathrm{L}(\mathcal{A}) = \mathrm{L}(q_I, \mathcal{A})$
- ⋆ two states p, q are equivalent in $\mathcal{A}$ if accepting runs coincide:

$$p \equiv_{\mathcal{A}} q \qquad :\Longleftrightarrow \qquad \mathrm{L}(p, \mathcal{A}) = \mathrm{L}(q, \mathcal{A})$$

- ⋆ merging equivalent states (e.g. $2 \equiv_{\mathcal{A}} 4$) does not change $L(\mathcal{A})$; results in minimal DFA

Table Filling Algorithm

Definition (Computing Distinguished States)

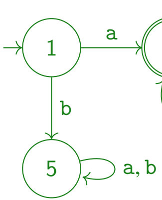
1. initially, we distinguish pairs $\mathcal{D} \triangleq \{(p, q) \mid p \in F \text{ and } q \notin F\}$
2. As long as new pairs are added, repeat:
   $\mathcal{D} := \mathcal{D} \cup \{(p, q) \mid \exists \mathtt{a} \in \Sigma.\ (\delta(p, \mathtt{a}), \delta(q, \mathtt{a})) \in \mathcal{D}\}$
3. Return $\mathcal{D}$

Table Filling Algorithm

Definition (Computing Distinguished States)

1. initially, we distinguish pairs $\mathcal{D} \triangleq \{(p, q) \mid p \in F \text{ and } q \notin F\}$
2. As long as new pairs are added, repeat:
   $\mathcal{D} := \mathcal{D} \cup \{(p, q) \mid \exists \mathtt{a} \in \Sigma.\ (\delta(p, \mathtt{a}), \delta(q, \mathtt{a})) \in \mathcal{D}\}$
3. Return $\mathcal{D}$

Example



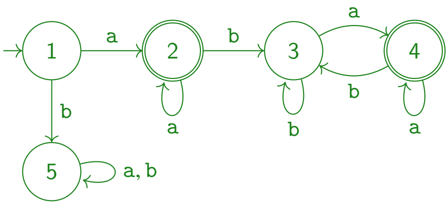
$\mathcal{D}$	1	2	3	4	5
1	—	—	—	—	—
2		—	—	—	—
3			—	—	—
4				—	—
5					—

Table Filling Algorithm

Definition (Computing Distinguished States)

1. initially, we distinguish pairs $\mathcal{D} \triangleq \{(p, q) \mid p \in F \text{ and } q \notin F\}$
2. As long as new pairs are added, repeat:
   $\mathcal{D} := \mathcal{D} \cup \{(p, q) \mid \exists \mathtt{a} \in \Sigma.\ (\delta(p, \mathtt{a}), \delta(q, \mathtt{a})) \in \mathcal{D}\}$
3. Return $\mathcal{D}$

Example



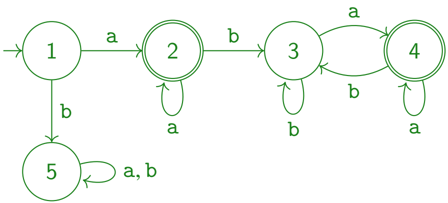
$\mathcal{D}$	1	2	3	4	5
1	—	—	—	—	—
2	∘	—	—	—	—
3		∘	—	—	—
4	∘		∘	—	—
5		∘		∘	—

Table Filling Algorithm

Definition (Computing Distinguished States)

1. initially, we distinguish pairs $\mathcal{D} \triangleq \{(p,q) \mid p \in F \text{ and } q \notin F\}$
2. As long as new pairs are added, repeat:
   $\mathcal{D} := \mathcal{D} \cup \{(p,q) \mid \exists \mathtt{a} \in \Sigma.\ (\delta(p,\mathtt{a}), \delta(q,\mathtt{a})) \in \mathcal{D}\}$
3. Return $\mathcal{D}$

Example



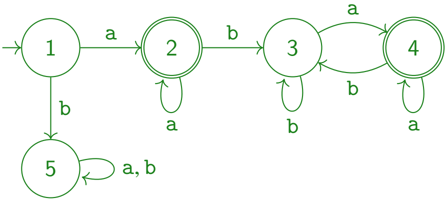
$\mathcal{D}$	1	2	3	4	5
1	—	—	—	—	—
2	∘	—	—	—	—
3		∘	—	—	—
4	∘		∘	—	—
5	∘	∘	∘	∘	—

Table Filling Algorithm

Definition (Computing Distinguished States)

1. initially, we distinguish pairs $\mathcal{D} \triangleq \{(p, q) \mid p \in F \text{ and } q \notin F\}$
2. As long as new pairs are added, repeat:
   $\mathcal{D} := \mathcal{D} \cup \{(p, q) \mid \exists \mathtt{a} \in \Sigma.\ (\delta(p, \mathtt{a}), \delta(q, \mathtt{a})) \in \mathcal{D}\}$
3. Return $\mathcal{D}$

Example



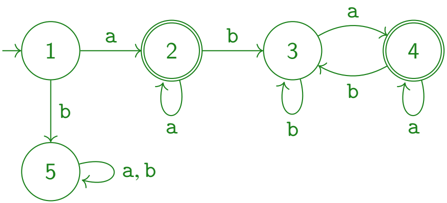
$\mathcal{D}$	1	2	3	4	5
1	—	—	—	—	—
2	∘	—	—	—	—
3	∘	∘	—	—	—
4	∘		∘	—	—
5	∘	∘	∘	∘	—

Table Filling Algorithm

Definition (Computing Distinguished States)

1. initially, we distinguish pairs $\mathcal{D} \triangleq \{(p, q) \mid p \in F \text{ and } q \notin F\}$
2. As long as new pairs are added, repeat:
   $\mathcal{D} := \mathcal{D} \cup \{(p, q) \mid \exists \mathtt{a} \in \Sigma.\ (\delta(p, \mathtt{a}), \delta(q, \mathtt{a})) \in \mathcal{D}\}$
3. Return $\mathcal{D}$

Example



$\mathcal{D}$	1	2	3	4	5
1	—	—	—	—	—
2	∘	—	—	—	—
3	∘	∘	—	—	—
4	∘		∘	—	—
5	∘	∘	∘	∘	—

Lemma (Correctness)

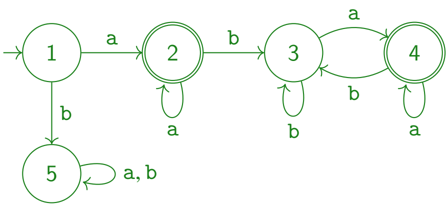
If two states are not distinguished, then they are equivalent.

Table Filling Algorithm

Definition (Computing Distinguished States)

1. initially, we distinguish pairs $\mathcal{D} \triangleq \{(p,q) \mid p \in F \text{ and } q \notin F\}$
2. As long as new pairs are added, repeat:
   $\mathcal{D} := \mathcal{D} \cup \{(p,q) \mid \exists \mathtt{a} \in \Sigma.\ (\delta(p,\mathtt{a}), \delta(q,\mathtt{a})) \in \mathcal{D}\}$
3. Return $\mathcal{D}$

Example



$\mathcal{D}$	1	2	3	4	5
1	—	—	—	—	—
2	∘	—	—	—	—
3	∘	∘	—	—	—
4	∘	$\equiv_{\mathcal{A}}$	∘	—	—
5	∘	∘	∘	∘	—

Lemma (Correctness)

If two states are not distinguished, then they are equivalent.

Minimisation

- let $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$ without non-reachable states (otherwise, remove them)
- note $\equiv_{\mathcal{A}}$ is an equivalence relation
- let $[q]$ denote the equivalence class of $q \in Q$
- define the quotient automata $\mathcal{A}_{\equiv} \triangleq (Q_{\equiv}, \Sigma, [q_I], \delta_{\equiv}, F_{\equiv})$ where:
  - $Q_{\equiv} \triangleq \{[q] \mid q \in Q\}$
  - $\delta_{\equiv}([q], \mathrm{a}) \triangleq [\delta(q, \mathrm{a})]$ for all $\mathrm{a} \in \Sigma$
  - $F_{\equiv} \triangleq \{[q] \mid q \in F\}$

Minimisation

- let $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$ without non-reachable states (otherwise, remove them)
- note $\equiv_{\mathcal{A}}$ is an equivalence relation
- let $[q]$ denote the equivalence class of $q \in Q$
- define the quotient automata $\mathcal{A}_{\equiv} \triangleq (Q_{\equiv}, \Sigma, [q_I], \delta_{\equiv}, F_{\equiv})$ where:
  - $Q_{\equiv} \triangleq \{[q] \mid q \in Q\}$
  - $\delta_{\equiv}([q], \mathrm{a}) \triangleq [\delta(q, \mathrm{a})]$ for all $\mathrm{a} \in \Sigma$
  - $F_{\equiv} \triangleq \{[q] \mid q \in F\}$

Theorem

The quotient automata $\mathcal{A}_{\equiv}$ is the minimal and unique DFA equivalent to $\mathcal{A}$

Discussion

How computationally difficult is it to ...

1. check $\mathsf{L}(\mathcal{A}) = \varnothing$ for given $\mathcal{A}$
2. check $w \in \mathsf{L}(\mathcal{A})$ for given $w \in \mathcal{A}$
3. check $\mathsf{L}(\mathcal{A}) = \Sigma^*$ for given $w \in \mathcal{A}$

Decision Procedures

Decision Problems

- A decision problem presents itself as a question to which must be answered yes or no.
  - Is the list sorted? Is the automaton minimal? etc.

Decision Problems

- ⋆ A decision problem presents itself as a question to which must be answered yes or no.
  - Is the list sorted? Is the automaton minimal? etc.
- ⋆ A decision problem depends on a given input, which has a certain size n
  - the list of length n, the automaton with n states, etc.

Decision Problems

- A decision problem presents itself as a question to which must be answered yes or no.
  - Is the list sorted? Is the automaton minimal? etc.
- A decision problem depends on a given input, which has a certain size n
  - the list of length n, the automaton with n states, etc.
- Often, a problem admits several algorithmic solutions, whose effectiveness varies.
- For some problems, no algorithmic solution exists
  - halting problem, Hilberts 10th problem, etc.

Decision Problems

- A decision problem presents itself as a question to which must be answered yes or no.
  - Is the list sorted? Is the automaton minimal? etc.
- A decision problem depends on a given input, which has a certain size n
  - the list of length n, the automaton with n states, etc.
- Often, a problem admits several algorithmic solutions, whose effectiveness varies.
- For some problems, no algorithmic solution exists
  - halting problem, Hilberts 10th problem, etc.
- To compare them, from a theoretical point of view, we usually assess their worst case complexity wrt. some notion of cost
  - e.g. time or space

Decision Problems

- A decision problem presents itself as a question to which must be answered yes or no.
  - Is the list sorted? Is the automaton minimal? etc.
- A decision problem depends on a given input, which has a certain size n
  - the list of length n, the automaton with n states, etc.
- Often, a problem admits several algorithmic solutions, whose effectiveness varies.
- For some problems, no algorithmic solution exists
  - halting problem, Hilberts 10th problem, etc.
- To compare them, from a theoretical point of view, we usually assess their worst case complexity wrt. some notion of cost
  - e.g. time or space
- The complexity is generally described by a function in the input size n.
- Usually, we are interested in an asymptotic analysis.
  - $O(n)$, $O(n^2)$, $O(2^n)$, ...

Complexity Classes

- The complexity of a problem can be thought of as the complexity of the best algorithm that solves it.

Complexity Classes

- The complexity of a problem can be thought of as the complexity of the best algorithm that solves it.

- this allows us to classify problems based on their inherent difficulty
  - polynomial time (P or PTIME), non-deterministic polynomial time (NP), exponential time (EXPTIME), etc.
  - polynomial space (PSPACE), etc.

Complexity Classes

- The complexity of a problem can be thought of as the complexity of the best algorithm that solves it.

- this allows us to classify problems based on their inherent difficulty
  - polynomial time (P or PTIME), non-deterministic polynomial time (NP), exponential time (EXPTIME), etc.
  - polynomial space (PSPACE), etc.

- complexity theory is concerned with the classification and relationships among classes

$$\mathrm{PTIME} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPTIME}$$

  - we know $\mathrm{PTIME} \subsetneq \mathrm{EXPTIME}$, but we do not know the status of individual inclusions
  - solving $\mathrm{PTIME} \overset{?}{\subsetneq} \mathrm{NP}$ is worth 1.000.000\$: a strict inclusion would separate, what we assume to be, feasible from unfeasible problems

Complexity Classes

- The complexity of a problem can be thought of as the complexity of the best algorithm that solves it.

- this allows us to classify problems based on their inherent difficulty
  - polynomial time (P or PTIME), non-deterministic polynomial time (NP), exponential time (EXPTIME), etc.
  - polynomial space (PSPACE), etc.

- complexity theory is concerned with the classification and relationships among classes

$$\mathrm{PTIME} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPTIME}$$

  - we know $\mathrm{PTIME} \subsetneq \mathrm{EXPTIME}$, but we do not know the status of individual inclusions
  - solving $\mathrm{PTIME} \overset{?}{\subsetneq} \mathrm{NP}$ is worth 1.000.000$: a strict inclusion would separate, what we assume to be, feasible from unfeasible problems
  - nowadays, some pretty good algorithms exists that can tackle unfeasible problems on average cases (e.g. **SAT** solvers)

The Word Problem

- Given: An NFA $\mathcal{A}$ with n states and word w of length $|w|$
- Question: $w \in \mathrm{L}(\mathcal{A})$?

Theorem

The word problem for NFAs is in PTIME.

The Word Problem

- Given: An NFA $\mathcal{A}$ with n states and word w of length $|w|$
- Question: $w \in L(\mathcal{A})$?

Theorem

The word problem for NFAs is in PTIME.

Proof Outline.

- the following depth-first search solves the problem in exponential time

```
def explore(q, w)
    if w is ϵ : return q ∈ F
    for p in δ(q, w[0]) :
        if explore(p, w[1:]) : return True
    return False
def member(w) : return explore(q_I, w)
```

The Word Problem

- Given: An NFA $\mathcal{A}$ with n states and word w of length $|w|$
- Question: $w \in L(\mathcal{A})$?

Theorem

The word problem for NFAs is in PTIME.

Proof Outline.

- the following depth-first search solves the problem in exponential time

```
def explore(q, w)
    if w is ϵ : return q ∈ F
    for p in δ(q, w[0]) :
        if explore(p, w[1:]) : return True
    return False
def member(w) : return explore(q_I, w)
```

- redundant calls can be eliminated via memoisation (i.e., tabulate calls explore(q, w))

The Word Problem

- Given: An NFA $\mathcal{A}$ with n states and word w of length $|w|$
- Question: $w \in L(\mathcal{A})$?

Theorem

The word problem for NFAs is in PTIME.

Proof Outline.

- the following depth-first search solves the problem in exponential time

```
def explore(q, w)
    if w is ϵ : return q ∈ F
    for p in δ(q, w[0]) :
        if explore(p, w[1:]) : return True
    return False
def member(w) : return explore(q_I, w)
```

- redundant calls can be eliminated via memoisation (i.e., tabulate calls explore(q, w))
- table bounded in size $O(n \cdot |w|^2)$

The Emptiness Problem

- Given: An NFA $\mathcal{A}$
- Question: $L(\mathcal{A}) = \varnothing$?

Theorem

The emptiness problem for NFAs is in PTIME.

The Emptiness Problem

- Given: An NFA $\mathcal{A}$
- Question: $L(\mathcal{A}) = \varnothing$?

Theorem

The emptiness problem for NFAs is in PTIME.

Proof Outline.

- essentially a graph reachability problem (why?)
- solvable by depth-first or breath-first search in time $O(n^2)$

The Universal Language Problem

- Given: An NFA $\mathcal{A}$
- Question: $L(\mathcal{A}) = \Sigma^*$?

Theorem

The universal language problem for NFAs is in PSPACE ⊆ EXPTIME.

- result non-trivial, because an infinity of words Σ^* should be accepting

The Universal Language Problem

- Given: An NFA $\mathcal{A}$
- Question: $L(\mathcal{A}) = \Sigma^*$?

Theorem

The universal language problem for NFAs is in PSPACE $\subseteq$ EXPTIME.

- result non-trivial, because an infinity of words Σ^* should be accepting
- however, the problem is equivalent to $\overline{L(\mathcal{A})} = \emptyset$

The Universal Language Problem

- Given: An NFA $\mathcal{A}$
- Question: $L(\mathcal{A}) = \Sigma^*$?

Theorem

The universal language problem for NFAs is in PSPACE $\subseteq$ EXPTIME.

- result non-trivial, because an infinity of words Σ^* should be accepting
- however, the problem is equivalent to $\overline{L(\mathcal{A})} = \emptyset$
- for DFAs, this amounts to checking $L(\overline{\mathcal{A}}) = \emptyset$, thus is in PTIME

The Universal Language Problem

- Given: An NFA $\mathcal{A}$
- Question: $L(\mathcal{A}) = \Sigma^*$?

Theorem

The universal language problem for NFAs is in PSPACE $\subseteq$ EXPTIME.

- result non-trivial, because an infinity of words Σ^* should be accepting
- however, the problem is equivalent to $\overline{L(\mathcal{A})} = \emptyset$
- for DFAs, this amounts to checking $L(\overline{\mathcal{A}}) = \emptyset$, thus is in PTIME
- translating NFAs to equivalent DFAs results in EXPTIME algorithm

The Universal Language Problem

- Given: An NFA $\mathcal{A}$
- Question: $L(\mathcal{A}) = \Sigma^*$?

Theorem

The universal language problem for NFAs is in PSPACE $\subseteq$ EXPTIME.

Proof Outline.

- we check $L(\mathcal{A}) = \Sigma^*$ in PSPACE for $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$

The Universal Language Problem

- Given: An NFA $\mathcal{A}$
- Question: $\mathrm{L}(\mathcal{A}) = \Sigma^*$?

Theorem

The universal language problem for NFAs is in PSPACE $\subseteq$ EXPTIME.

Proof Outline.

- we check $\mathrm{L}(\mathcal{A}) = \Sigma^*$ in PSPACE for $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$
- as we saw, this amount to translating $\mathcal{A}$ into an equivalent DFA $\mathcal{B}$ and checking $\overline{\mathcal{B}} = \varnothing$

The Universal Language Problem

- ⋆ Given: An NFA $\mathcal{A}$
- ⋆ Question: $L(\mathcal{A}) = \Sigma^*$?

Theorem

The universal language problem for NFAs is in PSPACE ⊆ EXPTIME*.*

Proof Outline.

- ⋆ we check $L(\mathcal{A}) = \Sigma^*$ in PSPACE for $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$
- ⋆ as we saw, this amount to translating $\mathcal{A}$ into an equivalent DFA $\mathcal{B}$ and checking $\overline{\mathcal{B}} = \varnothing$
- ⋆ constructing $\overline{\mathcal{B}}$ on-the-fly, this can be done non-deterministically in polynomial space

The Universal Language Problem

- Given: An NFA $\mathcal{A}$
- Question: $\mathrm{L}(\mathcal{A}) = \Sigma^*$?

Theorem

The universal language problem for NFAs is in PSPACE $\subseteq$ EXPTIME.

Proof Outline.

- we check $\mathrm{L}(\mathcal{A}) = \Sigma^*$ in PSPACE for $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$
- as we saw, this amount to translating $\mathcal{A}$ into an equivalent DFA $\mathcal{B}$ and checking $\overline{\mathcal{B}} = \emptyset$
- constructing $\overline{\mathcal{B}}$ on-the-fly, this can be done non-deterministically in polynomial space
- by Savich's theorem, any such algorithm can be turned into a deterministic one in PSPACE

Further Consequences

The Inclusion Problem

- Given: two NFA $\mathcal{A}$ and $\mathcal{B}$
- Question: $L(\mathcal{A}) \subseteq L(\mathcal{B})$?

The Equivalence Problem

- Given: two NFA $\mathcal{A}$ and $\mathcal{B}$
- Question: $L(\mathcal{A}) = L(\mathcal{B})$?

Theorem

Both problem are PSPACE *complete.*

- model checking, i.e., checking an implementation against high-level specifications, usually expressed as language inclusion.

Summary

	Word	Emptiness	Universality	Inclusion	Equivalence
DFA	PTIME	PTIME	PTIME	PTIME	PTIME
NFA	PTIME	PTIME	PSPACE	PSPACE	PSPACE

- Michael Rabin and Dana Scott received their Turing Award for their work "*Finite Automat and Their Decision Problems*"

Summary

	Word	Emptiness	Universality	Inclusion	Equivalence
DFA	PTIME	PTIME	PTIME	PTIME	PTIME
NFA	PTIME	PTIME	PSPACE	PSPACE	PSPACE

- Michael Rabin and Dana Scott received their Turing Award for their work "*Finite Automat and Their Decision Problems*"

Applications

- finite state machines (and its extensions) used in many disciplines
- efficient string search (Knuth-Morris-Pratt algorithm), e.g., in `grep`, `sed`, `awk`, `Java`, `C#`...
- Antivirus software
- DNA/protein analysis
- ...
- effectively satisiability/validity decision procedures for certain logics (see next lecture)

Programming Project (I)

Program a function $\texttt{match}(w, e)$ that matches a word w over alphabet $\Sigma = \{\mathrm{a}, \ldots, \mathrm{z}\}$ against a regular expression e

- regular expressions should encompass letters, union $e \mid f$, concatenation $e.f$ and e^*
  - bonus: complement, intersection, etc.
- test your implementation against Exercise 1

Programming Project (I)

Program a function $\texttt{match}(w, e)$ that matches a word w over alphabet $\Sigma = \{\texttt{a}, \ldots, \texttt{z}\}$ against a regular expression e

- regular expressions should encompass letters, union $e \mid f$, concatenation $e.f$ and e^*
  - bonus: complement, intersection, etc.
- test your implementation against Exercise 1
- concrete method and programming language up to you
- parser and stand-alone executable nice to have, but not a must

Programming Project (I)

Program a function $\texttt{match}(w, e)$ that matches a word w over alphabet $\Sigma = \{\mathrm{a}, \dots, \mathrm{z}\}$ against a regular expression e

- regular expressions should encompass letters, union $e \mid f$, concatenation $e.f$ and e^*
  - bonus: complement, intersection, etc.
- test your implementation against Exercise 1
- concrete method and programming language up to you
- parser and stand-alone executable nice to have, but not a must
- send solutions including instructions to `martin.avanzini@inria.fr`
- **deadline** Friday 23/04 08:00, exercise will be discussed in lecture 4