Theoretical Computer Science

Finite Automata

Technische Hochschule Rosenheim
Sommer 2022
Prof. Dr. Jochen Schmidt

Overview

Technische Hochschule Rosenheim

- Definition and representation of finite automata
- Recognized language of accepting automata
- Deterministic & nondeterministic automata
- Minimal automata

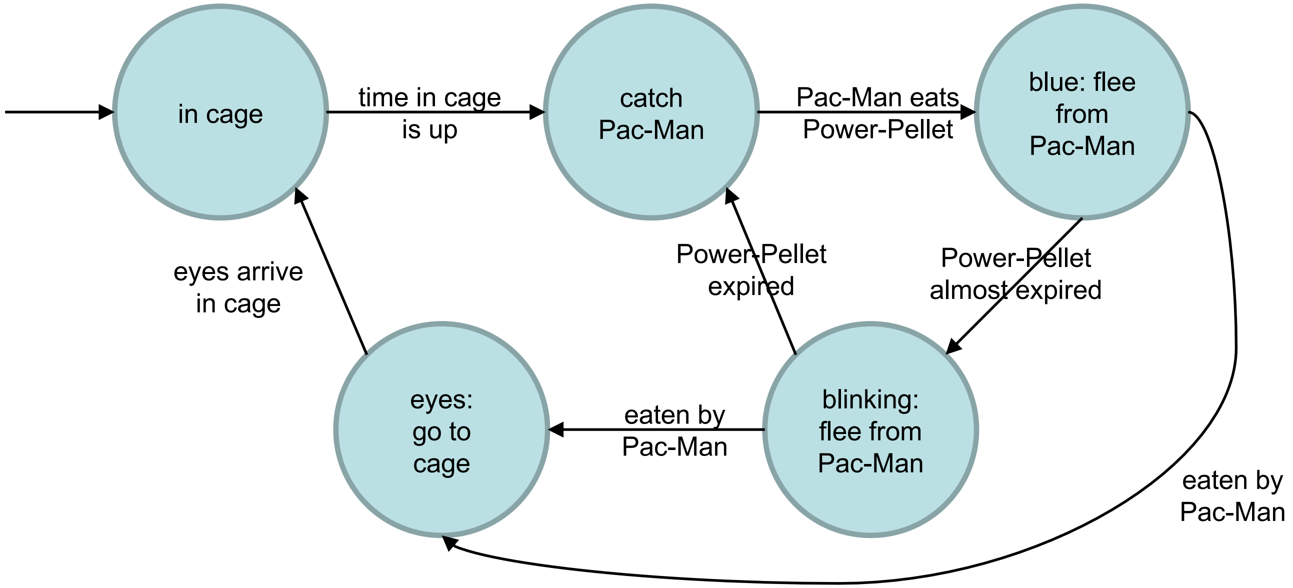
Automata

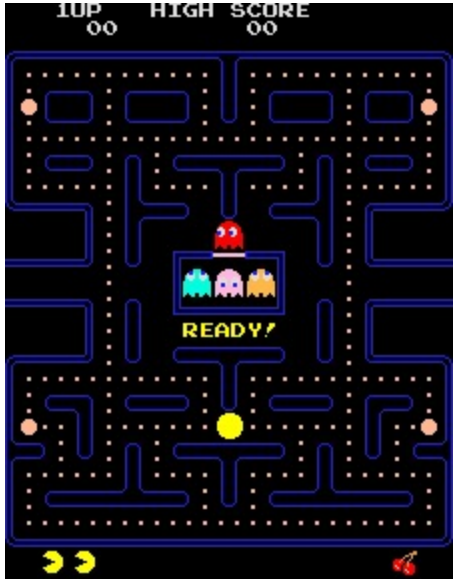
- Automaton in layman's terms:
  A machine that can control its behavior to a certain degree
  - e.g., a coffee dispenser (in German: Kaffee**automat**)
  - for applications in science and technology:
    a more mathematical, precise definition/abstraction is required

- Applications in computer science and related fields
  - analysis and formal representation of complex dependencies and processes
    - used for modelling in software engineering
    - define states (*Zustände*) and transitions between states
  - close connection to computability theory and formal languages
    - and therefore a part of the fundamentals of programming languages and compilers
  - electrical engineering: design of integrated circuits
    - minimization of number of states and logical gates
    - optimization of connections

Modelling Example: Pac-Man

- Model states and transitions using a deterministic finite automaton
- very useful & clear representation
  - which state transitions are allowed and when?
  - many algorithms available for finite automata
- Example: Ghosts in Pac-Man





Pac Man, Image: Namco

Deterministic Automaton

A deterministic automaton (*deterministischer Automat*) is defined by:

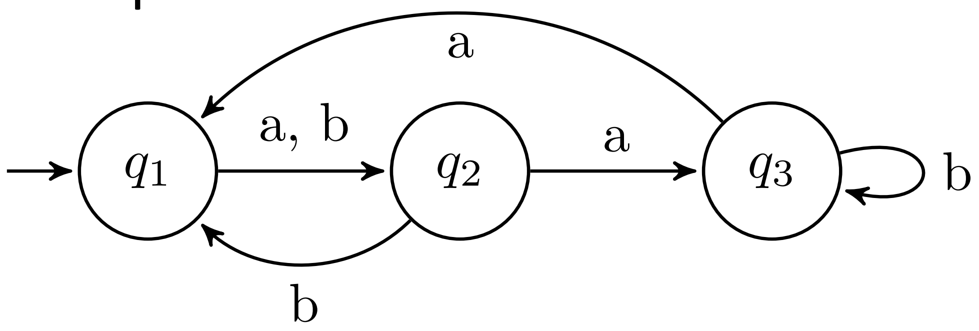
- A countable set (*abzählbare Menge*) of states (*Zustände*) $Q = \{q_1, q_2, \dots\}$
- An Alphabet (i.e., a countable, ordered set) of input symbols $\Sigma = \{\sigma_1, \sigma_2, \dots\}$
- A transition function (*Zustandsübergangsfunktion*), mapping state-input pairs to the next state $\delta: Q \times \Sigma \to Q$, where $Q \times \Sigma$ is the cartesian product of two sets, i.e., the set of all ordered pairs (q, σ) with $q \in Q$ and $\sigma \in \Sigma$
  - written as: $\delta(q_i, \sigma_j) = q_k$ or $q_i \sigma_j \to q_k$
  - successor state is unique (therefore: deterministic)
  - but not necessarily invertible
- A start state $q_s \in Q$
- If Q and Σ are finite sets: deterministic finite automaton (DFA)
  - other terms: finite-state machine or just state machine

State Diagrams & Transition Tables

- State diagrams are directed graphs:
  - vertices = states
  - edges = transitions
  - outdegree of each vertex = #symbols in input alphabet
  - start state is marked by an ingoing arrow

- Transition tables
  - in a DFA, the number of possible transitions is limited to: nz (n: #states, z: #symbols)
  - we can use a finite table to define the transition function

Example:



σ_i	q_1	q_2	q_3
a	q_2	q_3	q_1
b	q_2	q_1	q_3

Exercise

Draw the state diagram of an automaton having the following transition table:

Alphabet = {x, y, +}

States = {s0, s1, s2, s3, s4}

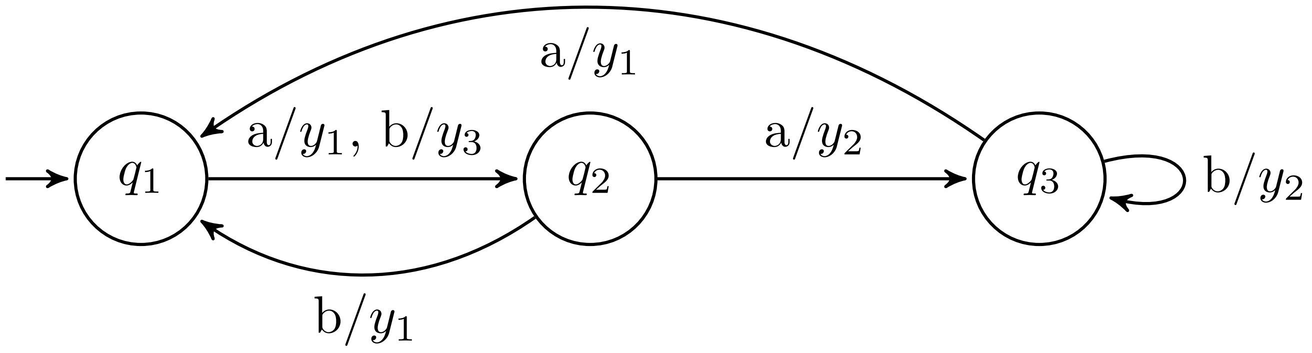
Start state = s0

	s0	**s1**	**s2**	**s3**	**s4**
x	s0	s3	s4	s3	s3
y	s0	s4	s1	s4	s4
+	s1	s2	s3	s4	s1

Automata with Output: Mealy Machine

- Extend definition by
  - an additional alphabet of output symbols $Y = \{y_1, y_2, \ldots\}$
  - an additional mapping g for generating the output

- Mealy machine – use $g: Q \times \Sigma \rightarrow Y$
  - output symbol depends on input symbol and current state

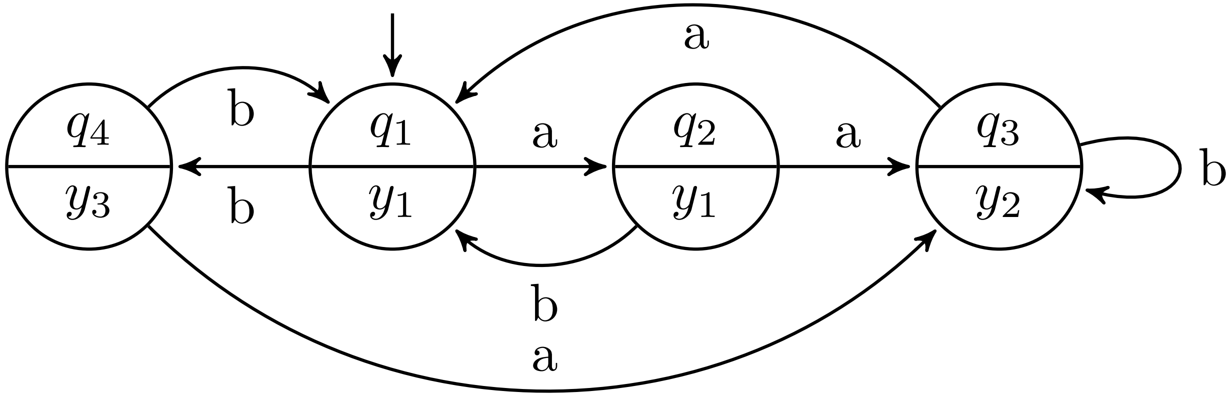


σ_i	q_1	q_2	q_3
a	q_2, y_1	q_3, y_2	q_1, y_1
b	q_2, y_3	q_1, y_1	q_3, y_2

Automata with Output: Moore Machine

- Extend definition by
  - an additional alphabet of output symbols $Y = \{y_1, y_2, \dots\}$
  - an additional mapping g for generating the output

- Moore machine – use $g: Q \to Y$
  - output symbol depends only on current state



σ_i	q_1	q_2	q_3	q_4
a	q_2, y_1	q_3, y_2	q_1, y_1	q_3, y_2
b	q_4, y_3	q_1, y_1	q_3, y_2	q_1, y_1

Automata with Output: Transducers

- Extend definition by
  - an additional alphabet of output symbols $Y = \{y_1, y_2, \ldots\}$
  - an additional mapping g for generating the output

- Mealy machine – use $g: Q \times \Sigma \to Y$
  - output symbol depends on input symbol and current state

- Moore machine – use $g: Q \to Y$
  - output symbol depends only on current state

- For each input sequence an output sequence is generated
- The automaton translates input to output $\longrightarrow$ transducer (*Transduktor, übersetzender Automat*)
- If Q, Σ, Y are finite sets: finite transduces (*endlicher Übersetzer*)
- Both definitions are equivalent – they can perform the same tasks

Multiple Inputs

Technische Hochschule Rosenheim

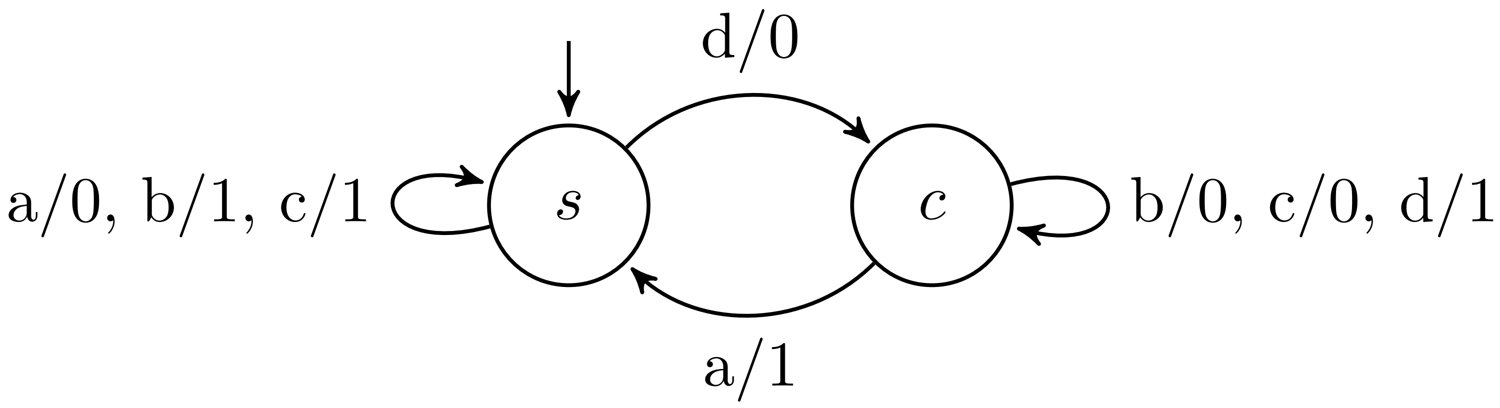
- Some problems require more than one input sequence
- By definition, an automaton can only process one input sequence

- Solution:
  - Redefine the combinations of symbols that appear simultaneously as new symbols of an extended alphabet
  - Construct an automaton that again processes only a single input sequence

Multiple Inputs – Example

Add two binary numbers

- original input alphabet: $\Sigma = \{0, 1\}$
- define new input alphabet $\Sigma' = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}\}$ with
  a: 0 + 0 = 0, b: 0 + 1 = 1, c: 1 + 0 = 1, d: 1 + 1 = 0 with carry 1
- output alphabet: $\mathrm{Y} = \{0, 1\}$ (result of adding to digits)
- states: carry (c) or no carry (s)

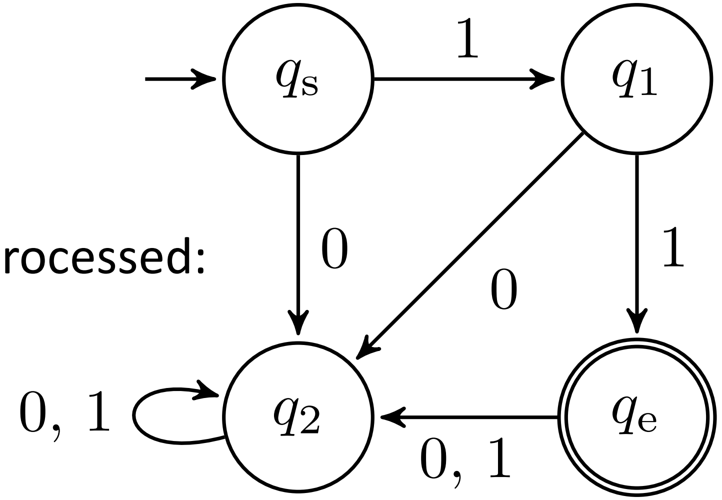


Acceptors: Recognized Language of Finite Automata

Accept States/Acceptor

- Again, we consider automata without output and extend the definition by:
- the set of accept states E (or end states, *akzeptierende Zustände bzw. Endzustände*)
    - $E \subseteq Q$, marked by two concentric circles in the state diagram

- This is an acceptor (*Akzeptor, erkennender Automat*)
    - Start processing of input string (left to right) at start state
    - Follow transitions for each symbol
    - If the automaton is in an end state after all symbols have been processed: The input string ("word") is said to be accepted.



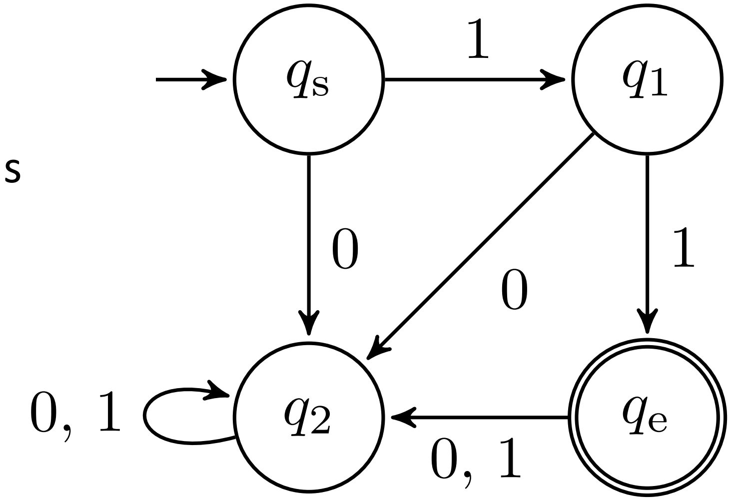
Recognized Language

- Set of all input words w accepted by an automaton A (i.e., it stops in an end state): recognized language $L(A)$ (*akzeptierte Sprache*)

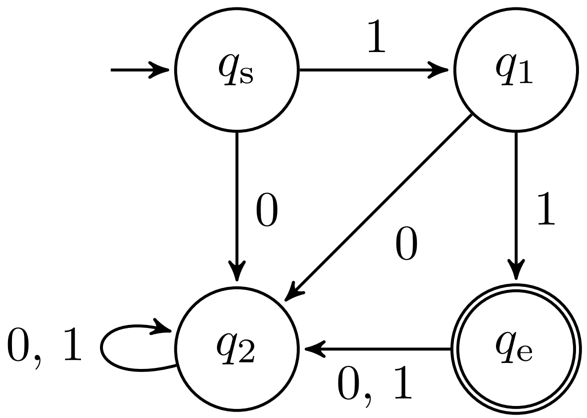
$$L(A) \coloneqq \{w \in \Sigma^* \mid (q_s, w) \to \cdots \to q_e, q_s = \text{start state}, q_e \in E\}$$

- Cardinality (*Mächtigkeit*) of L: $|L|$ = number of words of the language
- Two automata are said to be equivalent if they recognize the same language

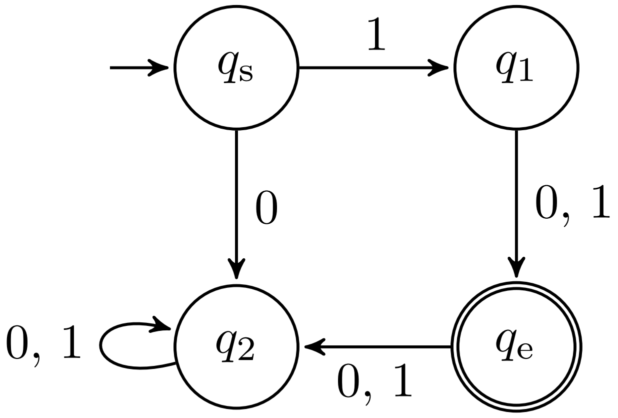
Recognized language contains only a single word:
$L = \{11\}, |L| = 1$



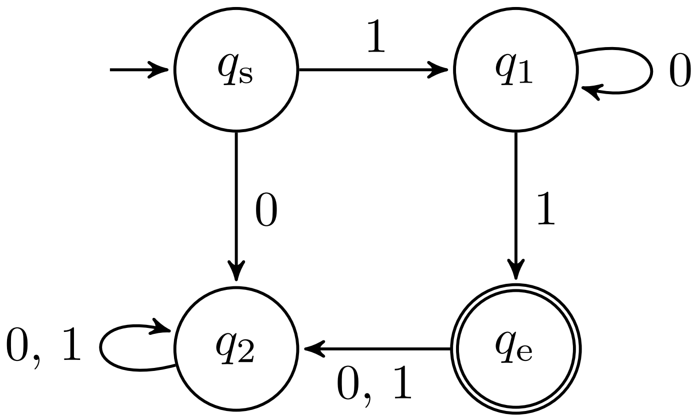
Recognized Language – Examples



$L = \{11\}, |L| = 1$



$L = \{11, 10\}, |L| = 2$



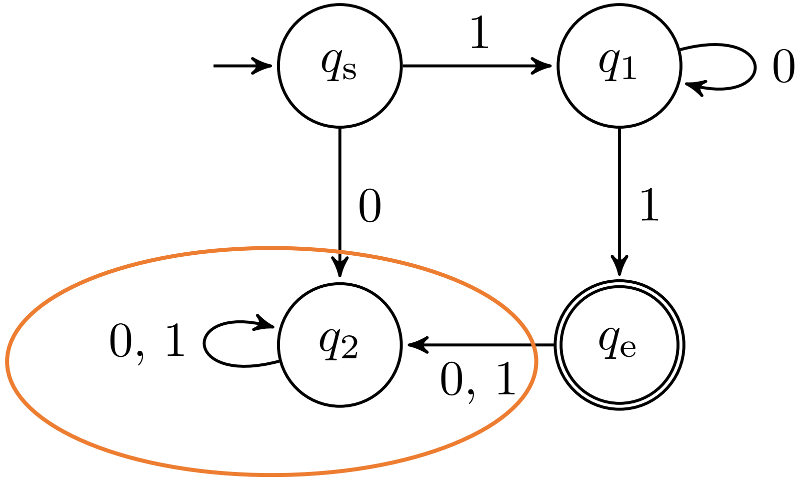
$L = \{10^n1 \mid n \in \mathbb{N}_0\}, |L| = \infty$

Finite Automata & Regular Languages

- The languages recognized by finite automata are identical to the regular languages
- Regular language: The subset of the word semigroup $(\Sigma^*, \circ)$ that can be generated by
  - string concatenation ($\circ$) = attach one string to another; symbol is usually omitted
  - set union ($\cup$)
  - Kleene closure (*)

- Kleene closure Σ^* of Σ (*Kleenesche Hülle*): $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$
  - contains all words that can be generated from the alphabet
  - including the empty string ε
- Kleene plus Σ^+ of Σ (*positive Hülle*): $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$
- ε is the neutral element of the semigroup
- Example: $\Sigma = \{a, b\}$
  - $\Sigma^* = \{\varepsilon, a, b, aa, bb, ab, ba, aaa, \dots\}$
  - $ab \circ abb = ababb,$ $(aa \circ bb) \circ aba = aa \circ (bb \circ aba) = aabbaba$

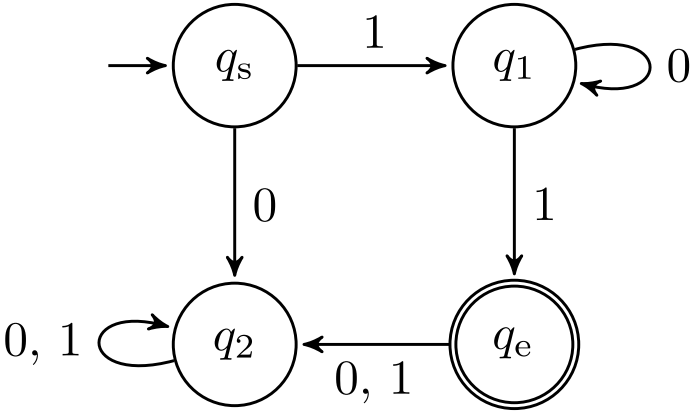
Trap States (*Fangzustände*)



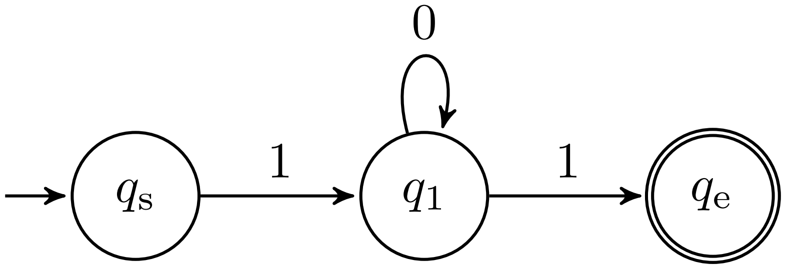
Trap state: Once reached, the automaton can never escape again – the processed word will not be part of the language, no matter which symbols follow. A trap state is **never** an end state!

These states are typically omitted for reasons of clarity $\longrightarrow$ incomplete automaton.

Incomplete Automata



complete automaton



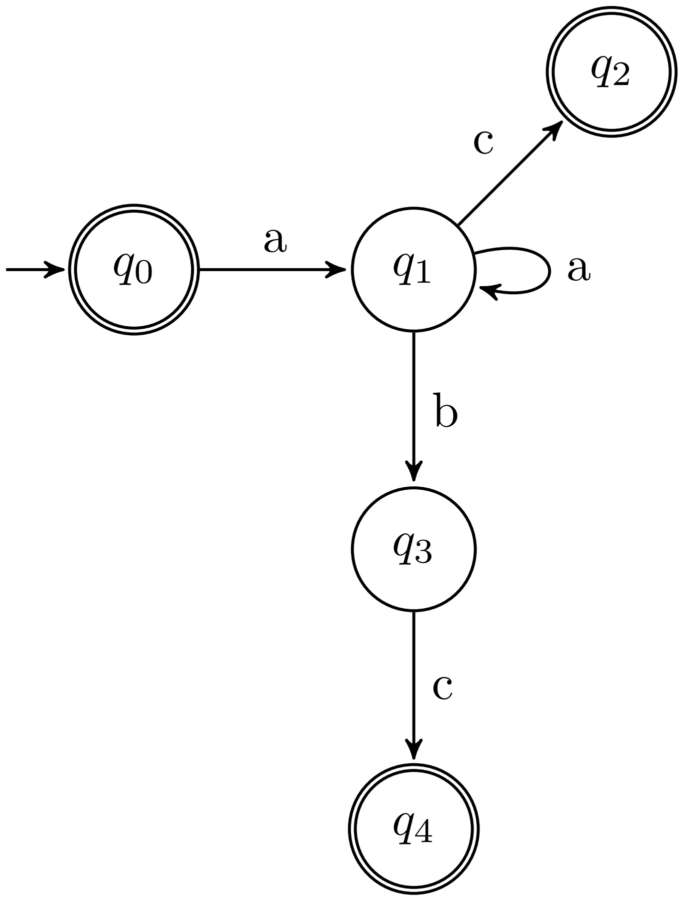
equivalent incomplete automaton

σ_i	q_s	q_1	q_e
0	—	q_1	—
1	q_1	q_e	—

- Trap states are usually omitted for clarity
- Convention: All transitions that are not shown end in a trap state
- In a compiler, these can be used to display error messages

Exercise

What is the recognized language of the following automaton?



$\mathbb{L} = \{\varepsilon, aa^n c, aa^n bc\}$ $\quad |\mathbb{L}| = \infty$

The Word Problem (*Wortproblem*)

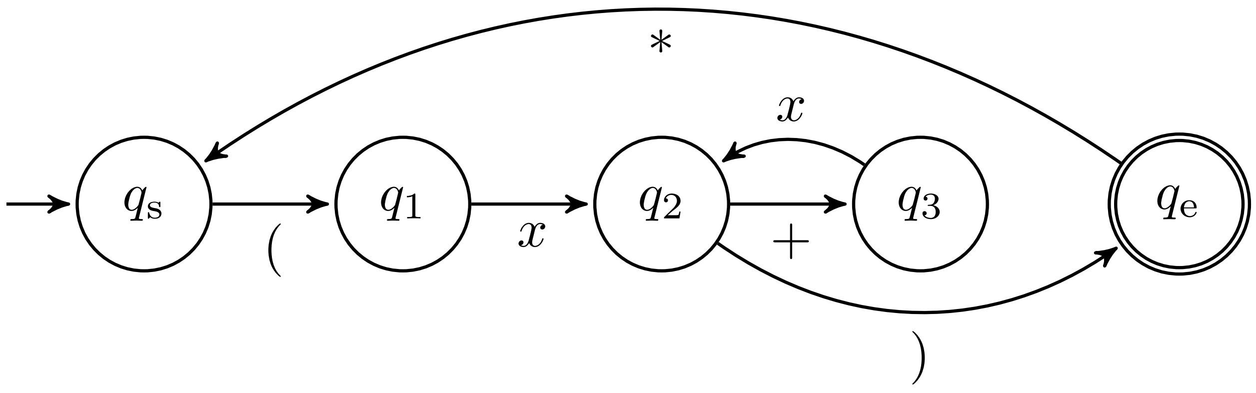
- Word Problem:
  Decide, whether an input string is part of the recognized language of an automaton.

- Application examples
  - Pattern matching: Finding a given string in a text
  - Lexical analysis in compilers
    - Decide, whether an input sequence (source code) adheres to the rules of a programming language
    - Split source code into elementary units (tokens), e.g., key words, identifiers, operators, …

Word Problem – Example

Analysis of simple arithmetic expressions with parentheses:

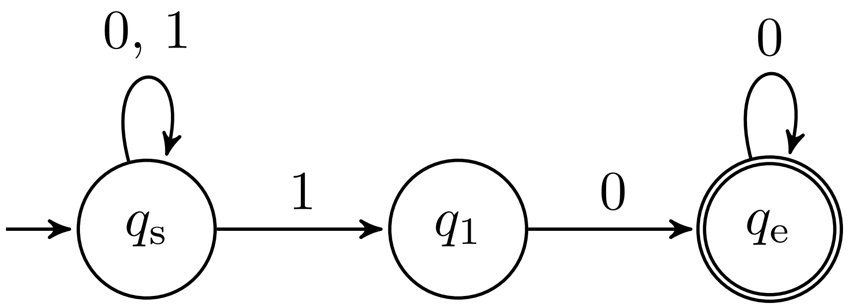
- Check, whether terms of the form (x+x)*(x+x+x) are built correctly
- Input alphabet: $\Sigma = \{(,\,),+,\ *,x\}$

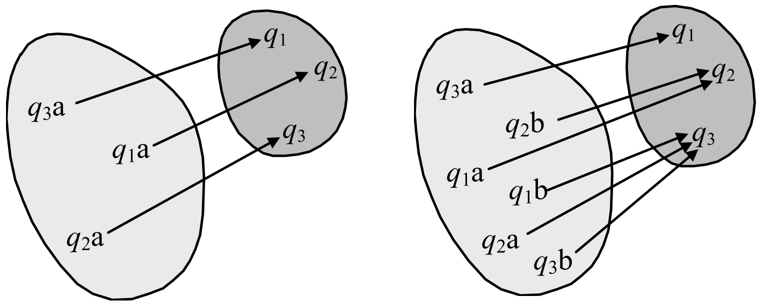


σ_i	q_s	q_1	q_2	q_3	q_e
$($	q_1	–	–	–	–
$)$	–	–	q_e	–	–
$+$	–	–	q_3	–	–
$*$	–	–	–	–	q_s
x	–	q_2	–	q_2	–

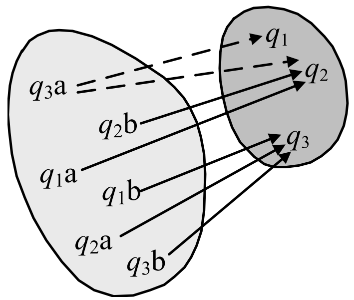
Nondeterministic Finite Automata (NFA)

- There is at least one state that has more than one transition with the same label
- Change the definition of the transition mapping from “function” to “relation”





deterministic

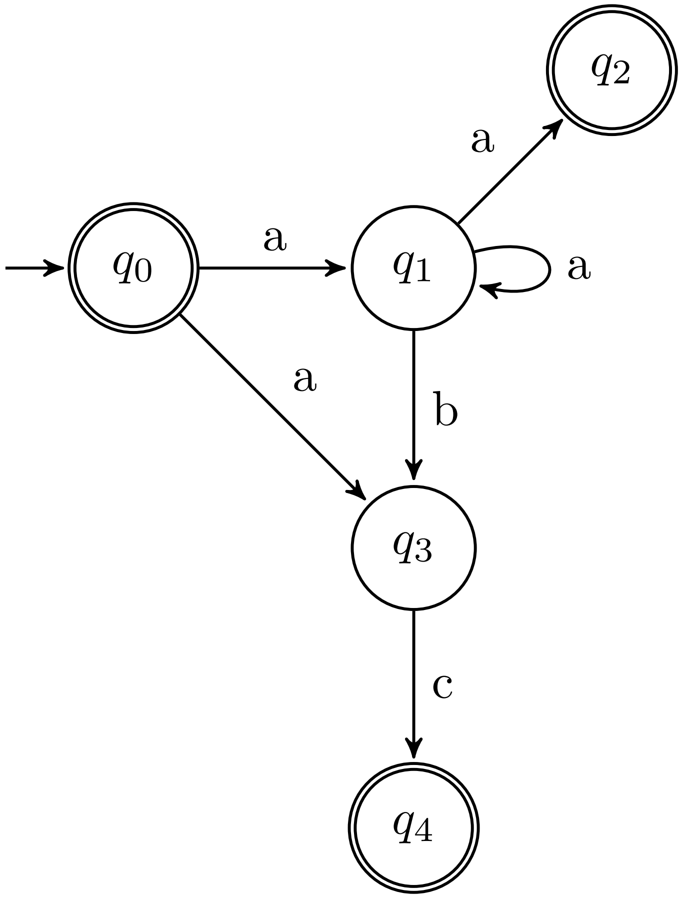


nondeterministic

- The automaton always chooses the correct successor state, without looking ahead (i.e., such that an end state will be reached, if it can be reached at all)
- Surprising: For any NFA we can construct an equivalent DFA
  - a deterministic version accepting the same language
  - nondeterminism does not result in more computational power (but may be faster)

Exercise

What is the recognized language of the following automaton?



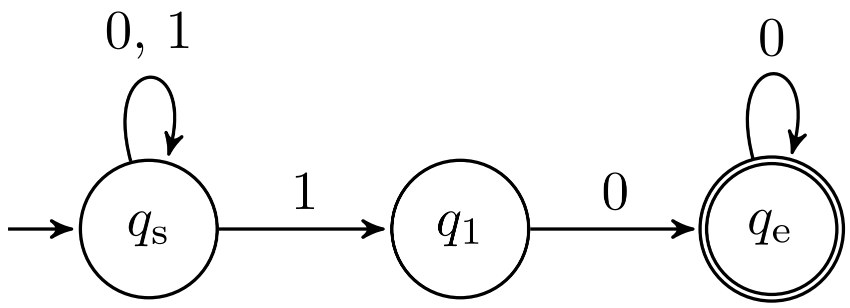
$L = \{\varepsilon, a^{2+n}, a^{1+n}bc, ac\}$ $|L| = \infty$

Transformation NFA ⟶ DFA

- For any nondeterministic finite acceptor (NFA), an equivalent deterministic one (DFA) can be constructed that recognizes the same language
- In 1976, M. Rabin and D. Scott received the Turing Award for the proof
- Basic idea:
  - given: NFA with n states
  - Determine all transitions of subsets of the state set Q
  - Set of all subsets: powerset (*Potenzmenge*); contains 2^n elements
  - the resulting DFA can thus have more states and transitions than the original NFA
  - in most cases, however, the number of states grows only slightly or even remains the same

Transformation NFA ⟶ DFA – Example

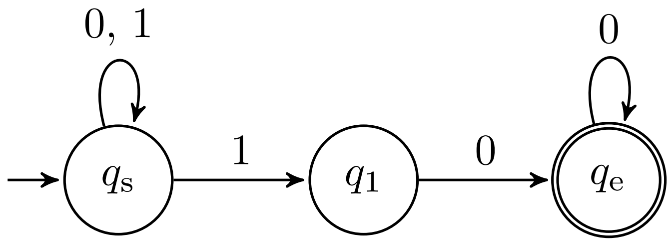
- given: NFA with $\Sigma = \{0,1\}$, $Q = \{q_s, q_1, q_e\}$
- recognized language: $L = \{x10^n \mid x \in \Sigma^*, n \in \mathbb{N}\}$



σ_i	q_s	q_1	q_e
0	q_s	q_e	q_e
1	q_s, q_1	—	—

Transformation NFA ⟶ DFA – Example

σ_i	q_s	q_1	q_e
0	q_s	q_e	q_e
1	q_s, q_1	–	–



(1) Start with all sets of successor states that can be reached directly from start state set $\{q_s\}$

σ_i	$\{q_\mathrm{s}\}$
0	$\{q_\mathrm{s}\}$
1	$\{q_\mathrm{s}, q_1\}$

(2) Add all newly listed subsets as new columns

σ_i	$\{q_\mathrm{s}\}$	$\{q_\mathrm{s}, q_1\}$
0	$\{q_\mathrm{s}\}$	$\{q_\mathrm{s}, q_\mathrm{e}\}$
1	$\{q_\mathrm{s}, q_1\}$	$\{q_\mathrm{s}, q_1\}$

(3) Iterate until no new columns must be added any more

σ_i	$\{q_\mathrm{s}\}$	$\{q_\mathrm{s}, q_1\}$	$\{q_\mathrm{s}, q_\mathrm{e}\}$
0	$\{q_\mathrm{s}\}$	$\{q_\mathrm{s}, q_\mathrm{e}\}$	$\{q_\mathrm{s}, q_\mathrm{e}\}$
1	$\{q_\mathrm{s}, q_1\}$	$\{q_\mathrm{s}, q_1\}$	$\{q_\mathrm{s}, q_1\}$

All subsets containing an end state of the NFA are end states of the DFA

Transformation NFA ⟶ DFA – Example

(4) Rename states (if desired)

σ_i	$\{q_s\}$	$\{q_s, q_1\}$	$\{q_s, q_e\}$
0	$\{q_s\}$	$\{q_s, q_e\}$	$\{q_s, q_e\}$
1	$\{q_s, q_1\}$	$\{q_s, q_1\}$	$\{q_s, q_1\}$

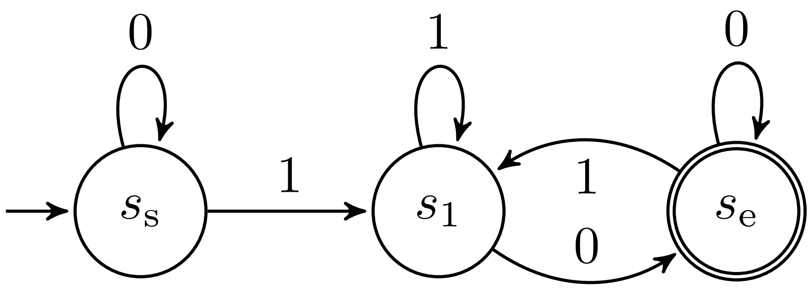
⟶

σ_i	s_s	s_1	s_e
0	s_s	s_e	s_e
1	s_1	s_1	s_1

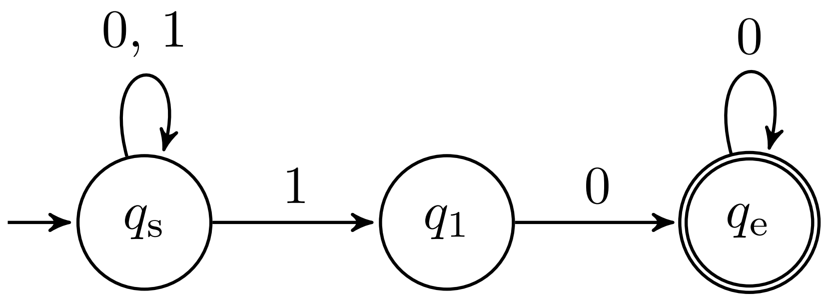
Out of a total of 8 possible states of the power set only 3 are actually used. Power set:

$$\wp(Q) = \{\{\ \}, \{q_s\}, \{q_1\}, \{q_e\}, \{q_s, q_1\}, \{q_s, q_e\}\{q_1, q_e\}, \{q_s, q_1, q_e\}\}$$

Resulting state diagram of DFA:



State diagram of original NFA:



$$L = \{x10^n \mid x \in \Sigma^*, n \in \mathbb{N}\}$$

Transformation NFA ⟶ DFA – Algorithm Summary

- Given: incomplete NFA
- Write all the input symbols in the first column of a table.
- Write the subset that contains all the initial states of the NFA as a header in the second column of the table. This is the initial state of the DFA.
- In the rows of the second column, enter the subsets that are reached from the initial state when the appropriate input symbol is entered.
- Add the new subsets in this column as new column headers to the table.
- In the rows of these columns, enter the subsets that are reached from the subsets in the column headers with the respective input symbols.
- Iterate until no new subsets/columns occur.
- The subsets found correspond to the states of the DFA.
- The transitions of the DFA result directly from the table entries.
- All subsets containing an end state of the NFA are end states of the DFA.
- For clarity, the state subsets can be renamed.

In any case, this algorithm stops after a finite number of steps, namely after a maximum of 2^n, since this corresponds to the cardinality of the power set.
⟶ Time complexity $O(2^n)$

ε-Moves

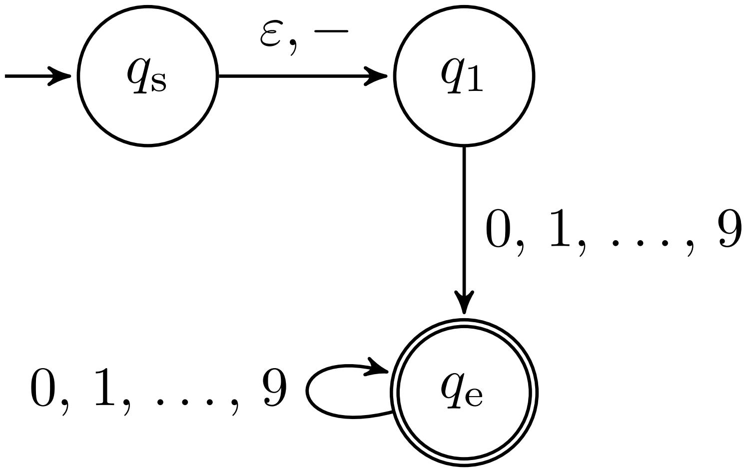
- DFA is deterministic: transition from the current state to the subsequent state is unambiguous
- DFA and NFA are also causal: a transition is determined by a specific input symbol

- Causality can be restricted: Allow spontaneous state transitions (not caused by input symbols)
- These are called ε-moves (or: λ-moves)
  - where ε is the empty string (of length 0)
  - ε is not an element of the input alphabet, but rather part of the transition function itself

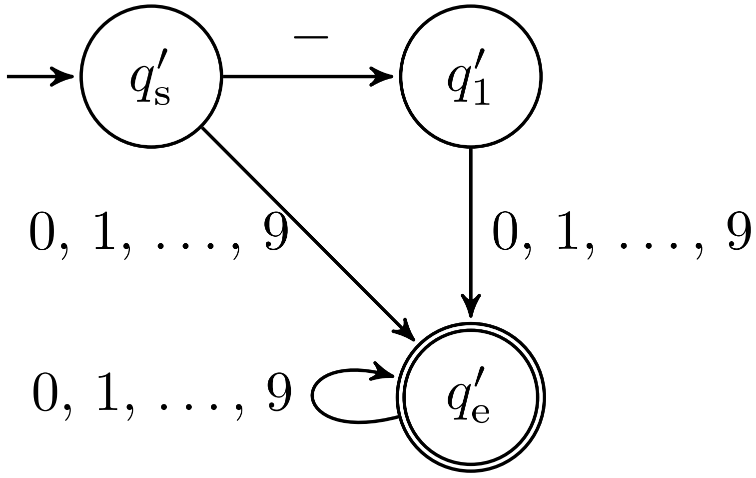
- **For any NFA with ε-moves, an equivalent DFA can be constructed containing no such transitions**
  - The contruction is the same as the Rabi-Scott algorithm for transforming NFA to DFA
  - with special treatment for ε-moves (use ε-closures, see textbooks for details)

ε-Moves – Example

Automaton recognizing integers, with an optional minus-sign



with ε-moves



without ε-moves

Minimal DFA

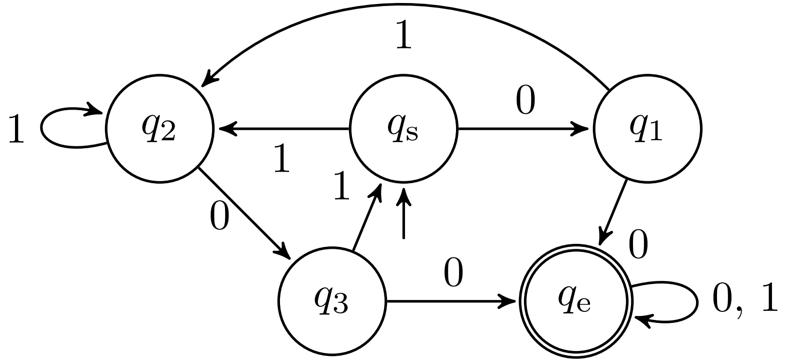
- The minimal DFA
  - is the one with the smallest number of states of all equivalent DFAs
  - is unique
  - can be constructed starting from any DFA (for NFA: convert to equivalent DFA first)

- Construction: Build equivalency classes (*Äquivalenzklassen*) by partitioning
  - Start with all states in a single partition
  - Separate end states from other states in a single partition
  - Split partitions until all states in a partition are equivalent

Minimal DFA – Example

Original transition table

σ_i	q_s	q_1	q_2	q_3	q_e
0	q_1	q_e	q_3	q_e	q_e
1	q_2	q_2	q_2	q_s	q_e



(1) Separate end states from other states

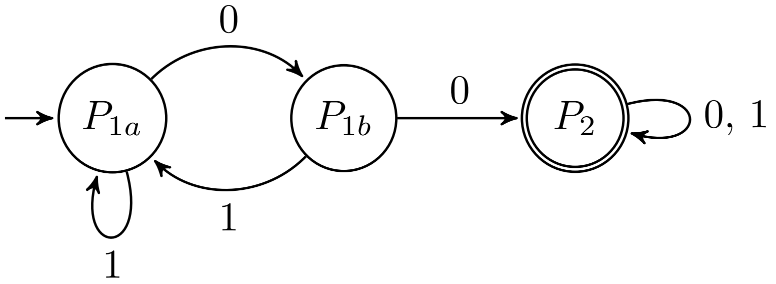
	P_1				P_2
σ_i	q_s	q_1	q_2	q_3	q_e
0	q_1, P_1	q_e, P_2	q_3, P_1	q_e, P_2	q_e, P_2
1	q_2, P_1	q_2, P_1	q_2, P_1	q_s, P_1	q_e, P_2

Result:

- all states of an equivalence class can be combined
- we get a DFA with 3 states:

(2) Split each partition until all states are equivalent

	P_{1a}		P_{1b}		P_2
σ_i	q_s	q_2	q_1	q_3	q_e
0	q_1, P_{1b}	q_3, P_{1b}	q_e, P_2	q_e, P_2	q_e, P_2
1	q_2, P_{1a}	q_2, P_{1a}	q_2, P_{1a}	q_s, P_{1a}	q_e, P_2



Summary

Technische Hochschule Rosenheim

- Automata with output: Both types are equivalent
  - Mealy: output at transition
  - Moore: output in state
- Accepting finite automata
  - process an input string (a “word”) from left to right
  - the word is accepted if the automata is in an end state after all symbols have been processed
  - the set of all accepted words is the recognized language
  - the recognized languages of finite automata are identical to the regular languages
  - incomplete automaton: trap states are omitted
- Deterministic (DFA) and nondeterministic (NFA) automata are equivalent
  - i.e., they have the same computational power (= they recognize regular languages)
  - an NFA may be faster though
  - ε-moves in NFAs allow for spontaneous transitions without cause
  - we can construct an equivalent DFA from any given NFA
  - we can construct an equivalent minimal DFA from any given DFA

Sources

- H. Ernst, J. Schmidt und G. Beneken: *Grundkurs Informatik*. Springer Vieweg, 7. Aufl., 2020.
- Schöning, U.: *Theoretische Informatik – kurz gefasst*. Spektrum Akad. Verlag (2008)
- Sander P., Stucky W., Herschel, R.: *Automaten, Sprachen, Berechenbarkeit*, B.G. Teubner, 1992