CS411 - Week 1, Intro

Clemens Kupke

20 January 2026

Attendance

Next

Hello!

Welcome to the class

- Welcome to Theory of Computation!
- Module leader email: `clemens.kupke@strath.ac.uk`
- Questions and Discussion on mattermost - details are on myplace
- class is assessed 70% exam, 30% coursework
- somewhat unusual: likely no coding required (it's theory after all)
- in principle there's ample of space to play with definitions and concepts by trying to implement them

Timetable

- a 2-hour lecture on Monday 10-12 (Assembly Hall)
- a 2-hour tutorial on Thursday 1-3pm (GH738)

What this class is about

- models of computation
- complexity of computation
- quite mathematical - definitions, theorems etc.
- please help me to keep it fun by being active in class!

Models of Computation

- allow us to define and study a class of problems (the problems computable by that model)
- we will:
  - introduce models (DFAs, Pushdown automata, Turing machines)
  - discuss how and what they compute
  - discuss what they don't compute!
- Crucially: we will formalise what is not computable.

We will then also use abstract models of computation in order to measure the complexity of a problem.

Next

First Model of Computation: Finite Automta

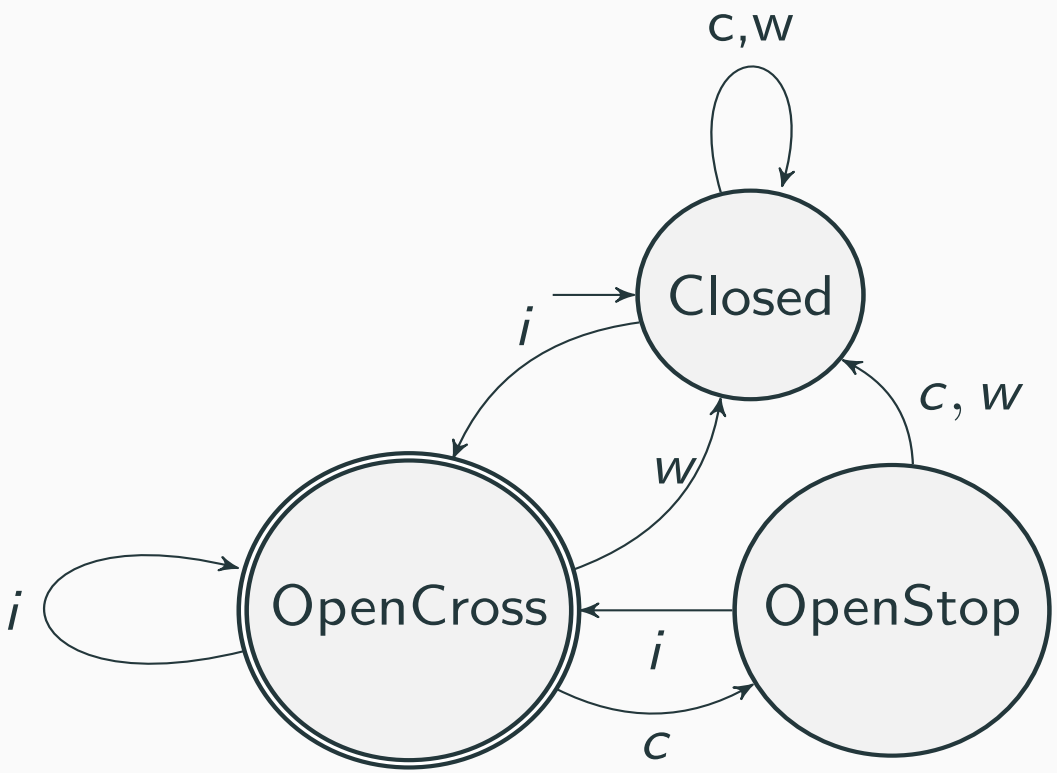
Finite Automata Intro

- Finite automata are a very intuitive model: a computer has a "state" which changes based on user input
- developed in the middle of the 20th century
- memory limited to the current state - this will be very different in the other models we will discuss
- extremely useful in many applications & still an active area of research

Example: Train Gate

- at many train stations you need to pass through a gate by inserting your ticket
- natural states: Gate closed (C), Gate open & possible to cross (OP), Gate open & impossible to cross (OI)
- actions are: insert (i), cross (c), wait(w)

Train Gate



We start at Closed, the state where the gate is closed (initial state).

The "good" sequences of actions are the one that end in the state where the gate is open and we are allowed to cross! $\Rightarrow$ The state OpenCross is the accepting state.

Sequences and Tuples

- A *sequence* is a collection of objects written in a certain order.
- Sequences are usually *written as a list within parentheses*.
- e.g. $(5, 2, E, W)$, (u, d, l, r).
- Finite sequences are often called *tuples*.
- A sequence with k elements is a k-tuple. A 2-tuple is called a pair.
- a word or string over an alphabet Σ is either the empty word ϵ or a finite list of elements of Σ
- we denote by Σ^* the set of all words over Σ
- a language (over alphabet Σ) is an arbitrary set $L \subseteq \Sigma^*$.

Problem $\sim$ Language

- Throughout this class we will take the perspective that a problem X can be encoded as a language L_X.

- solving the problem then boils down to be able to check membership in the language L_X.

- This will become clearer when we look at more examples - for now it's good to keep it in mind.

Formal Definition

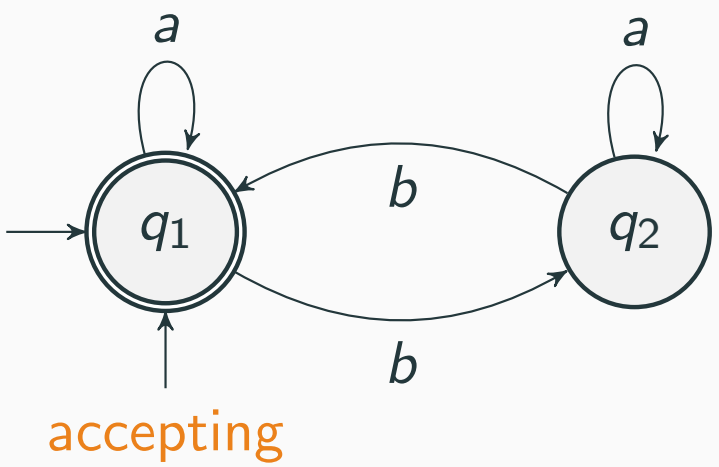
Definition (1.5)

A *deterministic finite automaton* (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of states
- Σ is a finite set ("alphabet")
- $\delta : Q \times \Sigma \to Q$ is the "transition function"
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept ("final") states

State Diagram & Transition Table

$\Sigma = \{a, b\},\ Q = \{q_1, q_2\},$



Transition Table:

δ	a	b
q_1	q_1	q_2
q_2	q_2	q_1

Acceptance (Intuitively)

- A DFA $(Q, \Sigma, \delta, q_0, F)$ starts at q_0 and processes a word $x = x_0 x_1 \dots x_n \in \Sigma^*$ "letter-by-letter" using the transition function δ

- for example it moves from q_0 to $\delta(q_0, x_0)$ etc

$$q_0 \to_{x_0} \delta(q_0, x_0) =: q_1 \to_{x_1} \delta(q_1, x_1) =: q_2 \to_{x_2} \dots$$

- if the automaton reaches an accepting state $q \in F$ after processing the given word x, then Q accepts x - otherwise it rejects x

Acceptance (formally)

Extend δ to words by recursion (induction on the length of a word).

$$\begin{aligned} \hat{\delta} : Q \times \Sigma^* &\to Q \\ \hat{\delta}(q, \epsilon) &:= q \\ \hat{\delta}(q, xa) &:= \delta(\hat{\delta}(q, x), a) \end{aligned}$$

Definition

A DFA $(Q, \Sigma, \delta, q_0, F)$ accepts a word x if $\hat{\delta}(q_0, x) \in F$.

Accepted Language, Regular Language

Definition

Let $(Q, \Sigma, \delta, q_0, F)$ be a DFA. The set

$$L = \{x \in \Sigma^* \mid (Q, \Sigma, \delta, q_0, F) \text{ accepts } x\}$$

is called language accepted by $(Q, \Sigma, \delta, q_0, F)$, language recognised by $(Q, \Sigma, \delta, q_0, F)$ or just language of $(Q, \Sigma, \delta, q_0, F)$.

Definition

Let Σ be a finite alphabet. A language $L \subseteq \Sigma^*$ is called *regular* if there exists a DFA $(Q, \Sigma, \delta, q_0, F)$ such that L is the language recognised by/accepted by $(Q, \Sigma, \delta, q_0, F)$.

Example I

$L = \{w \in \{a, b\}^* \mid \text{number of } a\text{'s is odd}\}$

Example II: substring

$L = \{w \in \{a, b\}^* \mid w \text{ contains abb } \}$

Example III: Formal Sums

$$L = \{w \in \{a, +, b, =, c\}^* \mid w \text{ is of the form } l_1 + l_2 + \cdots + l_n = l_{n+1} \text{ for some } l_i \in \{a, b, c\}, n \in \mathbb{N}\}$$

Next

Closure properties

Idea

- under which operations is the class of regular languages closed?
- understand which languages are regular
- understand the ("algebraic") structure of regular languages

$\Rightarrow$ establish a close connection between regular languages and so-called regular expressions

Basic regular languages (fixed alphabet Σ)

- empty language $L = \emptyset$
- all words $L = \Sigma^*$
- any finite language

Simple closure property: complement

Closure I: union

Theorem (1.45)

Let L_1 and L_2 be regular languages. Then $L_1 \cup L_2$ is a regular language.

Proof idea

Let $(Q_1, \Sigma, \delta_1, q_0^1, F_1)$ and $(Q_2, \Sigma, \delta_2, q_0^2, F_2)$ be the DFAs accepting L_1 and L_2. Then

$$(Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, (q_0^1, q_0^2), F)$$

with

$$(\delta_1 \times \delta_2)((q, q'), a) := (\delta_1(q, a), \delta_2(q', a))$$

and

$$F = \{(q, q') \in Q_1 \times Q_2 \mid q \in F_1 \text{ or } q' \in F_2\}$$

accepts $L_1 \cup L_2$.

Closure II: intersection

Same as union but with

$$F = \{(q, q') \in Q_1 \times Q_2 \mid q \in F_1 \text{ and } q' \in F_2\}$$

Closure III: composition

Definition

$L_1 \circ L_2 = \{xy \in \Sigma^* \mid x \in L_1, y \in L_2\}$

Theorem

Let L_1 and L_2 be regular languages. Then $L_1 \circ L_2$ is a regular language.

- Idea for automaton accepting $L_1 \circ L_2$: run automata for L_1 on the first part and "jump" to L_2 on the second part of a given word
- Problem: first and second part of a given word are not defined and automata cannot tell when it is finished examining the first part
- Jump has to occur *nondeterministically* - we need non-deterministic automata and the ability to jump ("ϵ transitions").

Next

NFAs

NFAs

- the next step of the computation is not determined by the current input
- machine can explore different routes
- computation succeeds if there exists a successful run
- feature: NFAs can be exponentially smaller than DFAs for the same language

Technical Detail: Powerset

Definition

For a set X we denote by $\mathcal{P}X$ the set of subsets of X and call $\mathcal{P}X$ the powerset of X.

Examples

$\mathcal{P}\emptyset = \{\emptyset\}$, $\mathcal{P}\{x\} = \{\emptyset, \{x\}\}$,
$\mathcal{P}\{x, y\} = \{\emptyset, \{x\}, \{y\}, \{x, y\}\}$,
$\mathcal{P}\{x, y, z\} = \{\emptyset, \{x\}, \{y\}, \{z\}, \{x, y\}, \{x, z\}, \{y, z\}, \{x, y, z\}\}$

Fun facts:

- $\#(\mathcal{P}X) = 2^{\#X}$ as subsets are in one-one correspondence with functions from X to 2 (How?)
- $\mathcal{P}X$ is a "Boolean Algebra"

(ϵ)-NFA: definition

Definition (1.37)

A nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ such that

1. Q is a finite set of states
2. Σ is a finite alphabet
3. $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}Q$ is the transition function
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

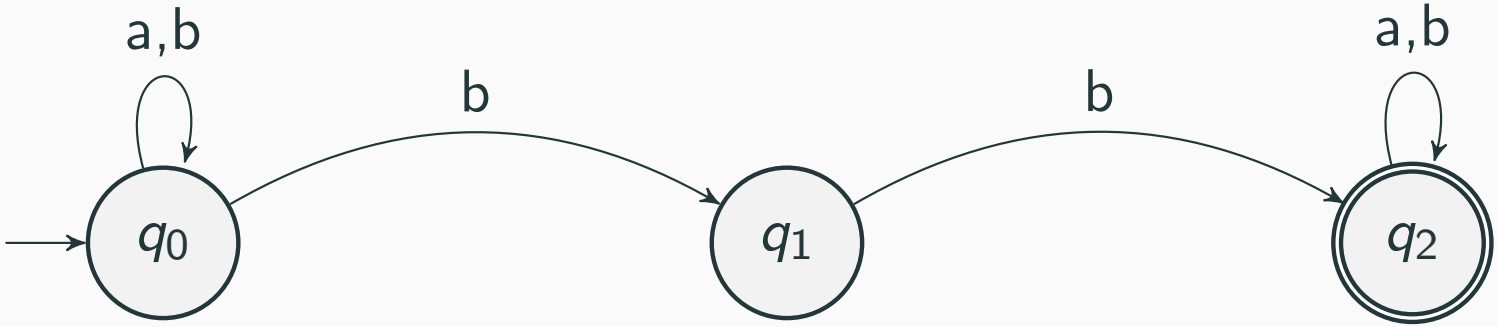
Here $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ where ϵ denotes the empty string (reading an empty string = "jumping").

NFA: acceptance

- Given NFA $(Q, \Sigma, \delta, q_0, F)$ and a word $w = x_0 x_1 \dots x_n$
- a run of the NFA on the word is obtained by selecting a successor
- for example we start at q_0 and then move to some $q \in \delta(q_0, x_0)$ (if this exist)
- a word w is accepted if the exists a run of the automaton on w that ends in a state in F
- as for DFAs we call the language L of all words accepted by $(Q, \Sigma, \delta, q_0, F)$ the language accepted by/recognised by it.

NFA: examples

Q_1:



$L_{Q_1} = \{w \mid w \text{ contains the string bb}\}$

NFA: examples

Next

NFAs are not more powerful than DFAs

From NFA to DFA: The Subset Construction

Let $(Q, \Sigma, \delta, q_0, F)$ be an NFA.

Definition

For a subset $U \subseteq Q$ we define its ϵ-closure

$$c(U) = \{q' \in Q \mid \exists q \in U . q' \in \delta(q, \epsilon)\}$$

Definition

We define a DFA $(Q', \Sigma, \Delta, q_0', F')$ equivalent to $(Q, \Sigma, \delta, q_0, F)$ by putting

- $Q' = \mathcal{P}(Q)$
- $\Delta(U, a) := c\left(\bigcup_{q \in U} \delta(q, a)\right)$
- $q_0' := c(\{q_0\})$
- $F' := \{U \mid F \cap U \neq \emptyset\}$

Equivalent automata?

- "equivalent" on the previous slide means "accepting the same language"

- we will discuss equivalence of automata a bit more next time

Example (contd)

Next

Closure Properties

Union revisted

Composition (now for real)

Regular expressions

Definition

The set REG of regular (Σ)-expressions is defined as follows:

1. for each $a \in \Sigma$, $a \in \mathrm{REG}$,
2. the empty word $\epsilon \in \mathrm{REG}$,
3. the empty set $\emptyset \in \mathrm{REG}$,
4. if $R_1, R_2 \in \mathrm{REG}$ then
   - $(R_1 \cup R_2) \in \mathrm{REG}$
   - $(R_1 \circ R_2) \in \mathrm{REG}$
5. if $R \in \mathrm{REG}$, then $R^* \in \mathrm{REG}$.

We usually write $(R_1 R_2)$ instead of $(R_1 \circ R_2)$ and $R_1 R_2 \ldots R_n$ for $((R_1 \circ R_2) \circ \ldots) \circ R_n)$

Note that the exact bracketing in the latter expression is not important as composition is associative.

Meaning of a regular expression

A regular expression R defines a language L_R as follows:

- $L(a) = \{a\}$ for $a \in \Sigma$
- $L(\epsilon) = \{\epsilon\}$
- $L(\emptyset) = \emptyset$
- $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$
- $L(R_1 \circ R_2) = L(R_1) \circ L(R_2)$
- $L(R^*) = (L(R))^*$

where $*$ means zero or finitely many repetitions:

$$L^* = \{w \mid w = \epsilon \text{ or } w = x_1 \dots x_n \text{ with } x_i \in L\}$$

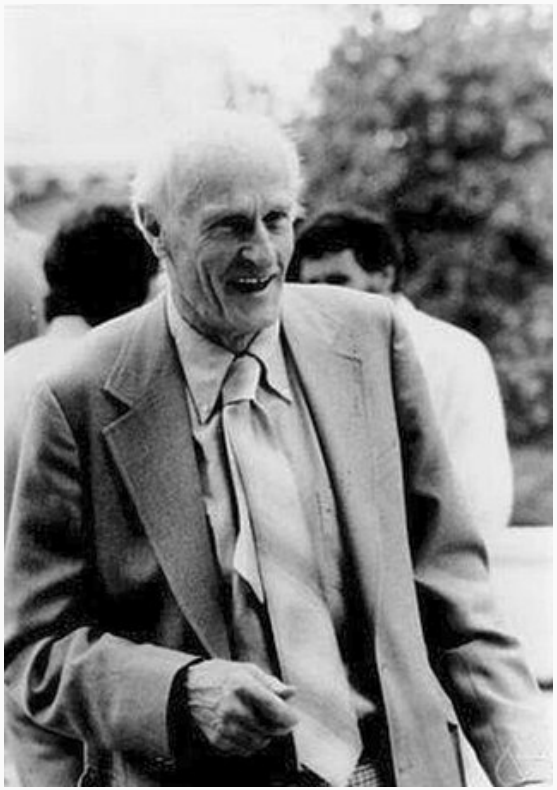
Some examples

Closure under $*$

Next

Equivalence Regular Expressions & regular languages

Kleene's Theorem



Stephen Cole Kleene

Photo by Konrad Jacobs, Erlangen, Copyright is MFO - Mathematisches Forschungsinstitut Oberwolfach,

https://commons.wikimedia.org/w/index.php?curid=12342617

Kleene's Theorem

Theorem

For a language $L \subseteq \Sigma^*$ the following are equivalent:

(i) L is represented by a regular expression

(ii) L is the language accepted by an NFA

(iii) L is the language accepted by a DFA

Proof Idea

(i) $\Rightarrow$ **(ii)** This is a consequences of the closure properties.

(ii) $\Rightarrow$ **(iii)** follows from language-preserving transformation NFA to DFA.

(iii) $\Rightarrow$ **(i)** Idea: turn DFA into so-called GNFA and reduce number of states to 2.

GNFA

Definition (1.64)

A *generalized non-deterministic finite automaton* (GNFA) is a 5-tuple $(Q, \Sigma, \delta, q_{\text{st}}, q_{\text{acc}})$ such that

1. Q is a finite set of states
2. Σ is a finite alphabet
3. $\delta : Q \setminus \{q_{\text{acc}}\} \times Q \setminus \{q_{\text{st}}\} \to \text{Reg}$
4. q_{st} is the start state
5. q_{acc} is the accept state

GNFA: Some intuitions

GNFA: acceptance

A GNFA $(Q, \Sigma, \delta, q_{\mathrm{st}}, q_{\mathrm{acc}})$ accepts a word $w \in \Sigma^*$ if $w = w_1 \dots w_n$ with $w_i \in \Sigma^*$ for $1 \leq i \leq n$ such that there exists a sequence of states $q_0 \dots q_n$ with

- $q_0 = q_{\mathrm{st}}$, $q_n = q_{\mathrm{acc}}$,
- for all i s.t. $1 \leq i \leq n$ we have

$$w_i \in L(R_i) \text{ for } R_i = \delta(q_{i-1}, q_i)$$

Compute a regular expression from GNFA

We sketch a *recursive* algorithm $\mathrm{Conv}(M)$ that converts a given GNFA M into an equivalent regular expression. Let k be the number of states of $M = (Q, \Sigma, \delta, q_{\mathrm{st}}, q_{\mathrm{acc}})$

1. if $k = 2$, then return $\delta(q_{\mathrm{st}}, q_{\mathrm{acc}})$
2. if $k > 2$, then remove arbitrary state q_{rip}, put $Q' = Q \setminus \{q_{\mathrm{rip}}\}$ and for all $q_i \in Q' \setminus \{q_{\mathrm{acc}}\}$ and $q_j \in Q' \setminus \{q_{\mathrm{st}}\}$ put

$$\delta'(q_i, q_j) = R_1(R_2)^* R_3 \cup R_4$$

   with $R_1 = \delta(q_i, q_{\mathrm{rip}})$, $R_2 = \delta(q_{\mathrm{rip}}, q_{\mathrm{rip}})$, $R_3 = \delta(q_{\mathrm{rip}}, q_j)$ and $R_4 = \delta(q_i, q_j)$. Let $M' = (Q', \Sigma, \delta', q_{\mathrm{st}}, q_{\mathrm{acc}})$ be the resulting GNFA.
3. return $\mathrm{Conv}(M')$ (recursive call!)

Example

Example

Finally proving (iii) $\Rightarrow$ (i)

To prove the implication of Kleene's theorem we start with a DFA (Q, Σ, δ, F) and turn it into the equivalent GNFA $(Q', \Sigma, \delta' : Q' \times \mathrm{REG} \to \mathcal{P}Q', F')$ given by putting

- $Q' = Q \cup \{q_{\mathrm{st}}, q_{\mathrm{acc}}\}$
- $\Delta(q, R) = \begin{cases} \{q_0\} & \text{if} \quad q = q_{\mathrm{st}} \text{ and } R = \epsilon \\ \{\delta(q, a)\} & \text{if} \quad q \in Q, R = a \in \Sigma \\ \{q_{\mathrm{acc}}\} & \text{if} \quad q \in F, R = \epsilon \\ \emptyset & \text{otherwise.} \end{cases}$
- $\delta'(q, q') := R \in \mathrm{REG} \quad \text{if } q' \in \Delta(q, R)$;
- $F' = \{q_{\mathrm{acc}}\}$

We then use the above algorithm Conv to compute the regular expression for Q' and thus for Q.