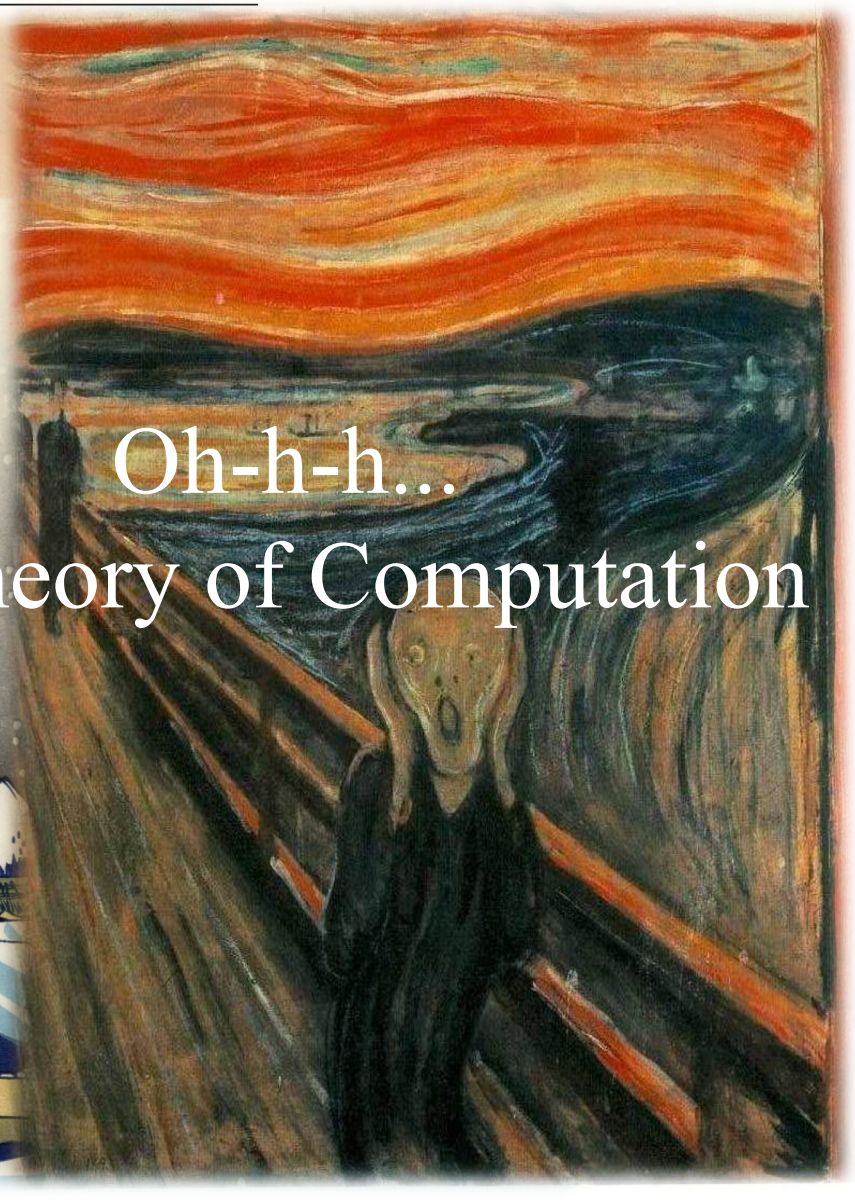
Università
della
Svizzera
italiana

**Facoltà
di scienze
informatiche**



Oh-h-h...
it's Theory of Computation

Today's lecture

- Computability theory

  - Decidability and decidable problems

Recall Hilbert and his 10th problem

- In 1900: Posed 23 "challenge problems" in Mathematics
- The 10th problem:
  Devise an algorithm to decide if a given polynomial has an integral root.
- We now know: This is undecidable!
  - Needed a definition of "algorithm", which was given by Church and Turing (independently)

Church-Turing Hypothesis

Church-Turing Hypothesis

- Intuitive notion of algorithms
= Turing machine algorithms

- "Any process which could be naturally called *an effective procedure* can be realized by a Turing machine"

TM and Algorithms

Turing model serves as a precise model for the definition of algorithm

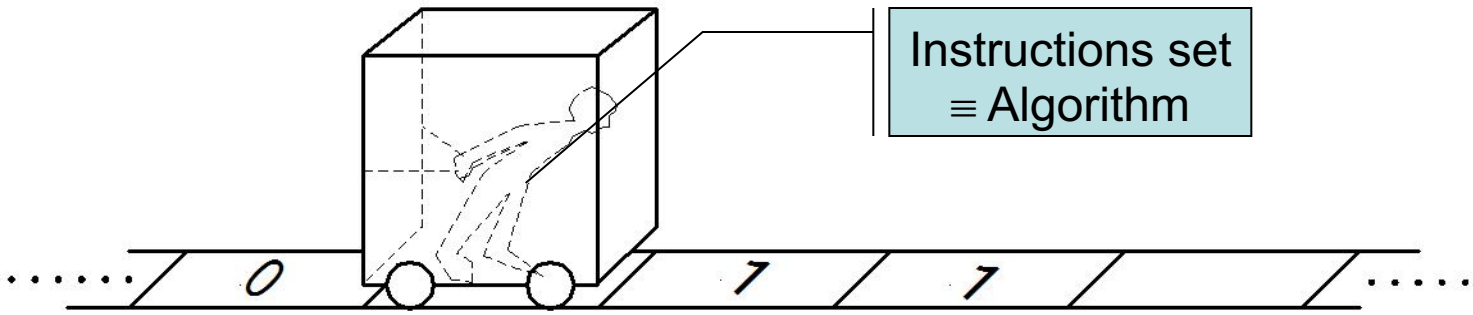
Computability

- What is computability about?
  - Can you describe it in terms of algorithms?

Università
della
Svizzera
italiana

Facoltà
di scienze
informatiche

Understanding Turing Machine and Algorithms



We (and ToC) study power of algorithms to solve problems

Target: to explore the limits of algorithmic solvability (computability)

Why study unsolvability?

Suggestions?

Realizing that problem can not be solved helps to restate/simplify the problem

Gives insight into perspective on computation/computer science

What does it mean that the problem is solvable?

- How can we find out whether a problem is solvable?

If we can find a Turing Machine that decides that problem then we know that the problem is solvable

Standardized way to describe algorithms

The input to Turing machine: string

If an input object is not a string, the object MUST be represented as a string

The first step of Turing Machine is to check if the input is correct – that is that it is a string

Example

A = { <G> | G is a connected undirected graph}

Here A is a language of all strings representing graphs that are connected

<G> - encoding of a graph into it's string representation

How G can be encoded?

Example

A = { <G> | G is a connected undirected graph}

Here is the *high-level description* of TM that decides A

TM = “on input <G>:

1. Select the 1st node and label it;
2. Repeat the following until no new vertices are labeled:

   For each node, label it if it is attached by an edge to a node that is already labeled;
3. Scan all nodes. If they are all labeled, accept.

   Otherwise, reject.”

For *implementation-level* details see example 3.23 in the book

Decidability of languages

- We will look at sample classes of decidable languages
  - Decidability of automata problems
  - Decidability of CFG problems

Importance in practice

- To what practical issue does the following problem relate to?
  - Testing if a string is a member of a CFL

It is related to the problem of recognizing and compiling programs in a programming language

Every time you are programming, you deal with this task: what is your intuition? Is it decidable?

YES, it is

Finite Automata problems

- Can you give examples of any algorithm related to finite automata?

- Common use algorithms for testing if

  – FA accepts a string (acceptance testing)
  – The language of a FA is empty (emptiness testing)
  – Two FAs are equivalent (equivalence test)

Notation

- We will represent various computational problems *by languages*
- For example, the acceptance problem for DFAs of testing if a particular DFA accepts a given string can be expressed as a language

$A_{DFA} = \{\langle B,\omega\rangle \mid B \text{ is a DFA that accepts string } \omega\}$

this language contains the encodings of all DFAs together with strings that the DFAs accept

Decidable problems

Showing that the *language is decidable* is the same as showing that the *computational problem is decidable*

DFA Acceptance test

- *Theorem*: A_{DFA} is a decidable language

- *Proof Idea:*
  - We need to identify a Turing Machine, M, that decides A_{DFA}

Suggestions?

DFA Acceptance test

- *Theorem*: A_{DFA} is a decidable language
- *Proof Idea:*
  - We need to identify a Turing Machine, M, that decides A_{DFA}

M = " On input $\langle B,\omega \rangle$, where B is a DFA and ω is a string:

1. Simulate B on ω
2. If the simulation ends in an accepting state, *accept*. Otherwise, *reject*."

DFA Acceptance test

Proof Ideas:

- How to keep track of changes in DFA?
- How to simulate DFA on an input string ?
- How to decide if DFA accepts the string?

DFA Acceptance test

Proof:

1. TM, M, Simulation (high-level description):

- For a DFA = $(Q, \Sigma, \delta, q_0, F)$ TM keeps track of the DFA's current state and its current position in ω by writing this information on the tape
- The states and the tape position are updated according to δ

2. When TM finishes with the last symbol of ω, M *accepts* ω if B is in the accepting state; otherwise it *rejects* it.

NFA Acceptance test

Theorem: A_{NFA} is a decidable language

What is A_{NFA}?

$A_{NFA} = \{\langle B,\omega\rangle \mid B \text{ is a NFA that accepts string } \omega\}$

NFA Acceptance test

Theorem: A_{NFA} is a decidable language

Proof Ideas?

Option 1: simulate an NFA instead of DFA in the previous proof

Option 2: Convert the NFA into a DFA and then use the same proof as for A_{DFA}

NFA Acceptance test

Option 2: Convert the NFA into a DFA and then use the same proof as for A_{DFA}

Important observation:

Conversions/operations (e.g., negation, intersections, etc.) are common techniques used in ToC: they help reducing the complexity of the proof by reusing the existing proofs. Use this approach!

NFA Acceptance test

Proof outline (for option 2):

N = " On input $\langle B,\omega \rangle$, where B is an NFA and ω is a string:

1. Convert B to an equivalent DFA C (recall a procedure for NFA to DFA procedure)
2. Simulate C on ω:

   Run M from DFA test on <C, ω>

3. If the simulation ends in an accepting state, *accept*. Otherwise, *reject*."

Problem from Reg. Expressions

What is the most common problem to address when it comes to regular expressions?

To determine if a regular expression generates a given string

Language of REX?

Problem from Reg. Expressions

What is the most common problem to address when it comes to regular expressions?

To determine if a regular expression generates a given string

Language of REX?

$A_{REX} = \{\langle R,\omega\rangle \mid R \text{ is a regular expression that generates string } \omega\}$

Problem from Reg. Expressions

Theorem: A_{REX} is a decidable language

$A_{REX} = \{\langle R,\omega\rangle \mid R$ is a regular expression that generates string $\omega\}$

Proof Ideas?

Problem from Reg. Expressions

Proof: The following TM decides A_{REX}

P = "On input $\langle R,\omega \rangle$:

1. Convert R to an equivalent NFA A
2. Run TM , N, from NFA test, on input $\langle A, \omega \rangle$
3. If N accepts, *accept.*

   Otherwise, *reject."*

Important Observation

For decidability purposes, presenting Turing Machines with DFAs, NFAs, or regular expressions are all equivalent because the Turing Machine is able to convert one form of encoding to another

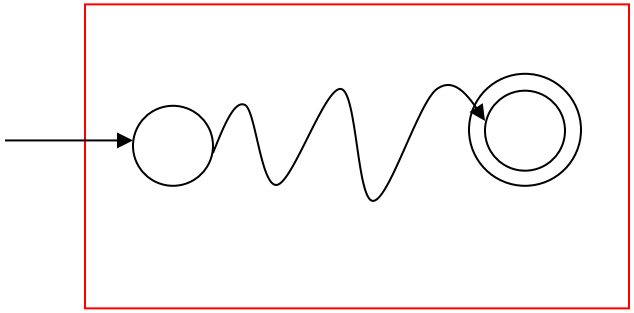
More Automata problems

Now let's turn to a different kind of problem: emptiness testing

What is the language to deal with?

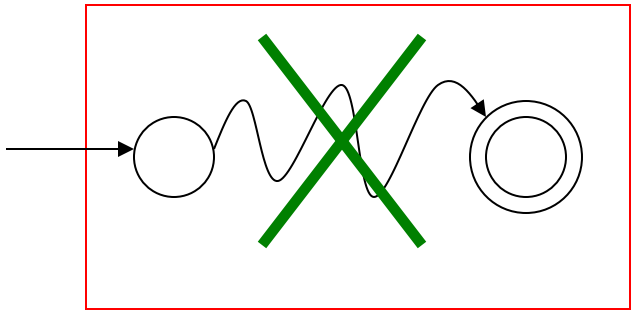
$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \varnothing\}$$

DFA



$L \neq \emptyset$

DFA



$L = \emptyset$

DFA emptiness test

Theorem: E_{DFA} is a decidable language

Proof Ideas?

Recall: A DFA accepts some string iff reaching an accept state from the start state by traveling along the arrows of the DFA is possible

Proof: to test the above
How?

DFA emptiness test

Theorem: E_{DFA} is a decidable language

Proof:

TM S = ‘’On input ⟨A⟩, where A is a DFA:

1. Mark the start state of A
2. Repeat until no new states get marked:

   Mark any state that has a transition coming into it from any state that is already marked

3. If *no accept state* is marked, *accept*; otherwise, *reject.*”

Example

Let A = {⟨M⟩ | M is a DFA which does not accept any string containing an odd number of 1s.} Show that A is decidable

Ideas?

Example

Let A = {⟨M⟩ | M is a DFA which does not accept any string containing an odd number of 1s.} Show that A is decidable

Here is the TM that decides A:

"On input <M>:
1. Construct a DFA N that accepts every string with an odd number of 1s;
2. Construct a DFA Q that $L(Q) = L(M) \cap L(N)$;
3. Test whether L(Q) is empty using decider (S) of E_{DFA}
4. If S accepts, accept; otherwise reject"