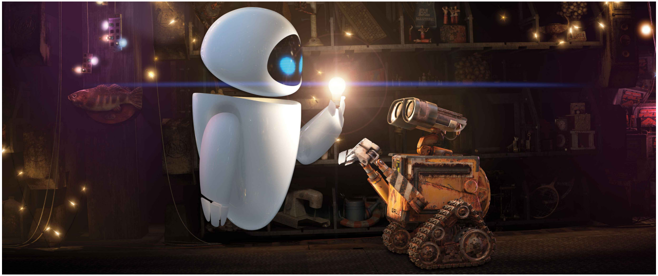
Zero Knowledge Proofs

MoL Research Project



Maaike Zwart & Suzanne van Wijk

January 2015

Institute for Logic, Language and Computation

Overview

In this project:

Lectures:

- Complexity Theory
  - Turing Machines
  - Classes P and NP
  - Interactive proofs
- Zero Knowledge Proofs
  - ZK proofs, intuitive
  - ZK proofs
  - ZK proofs for NP problems
  - Commitment scemes

Overview

In this project:

Lectures:

- Complexity Theory
  - **Turing Machines**
  - **Classes P and NP**
  - Interactive proofs
- Zero Knowledge Proofs
  - ZK proofs, intuitive
  - ZK proofs
  - ZK proofs for NP problems
  - Commitment scemes

Overview

In this project:

Lectures:

- Complexity Theory
  - Turing Machines
  - Classes P and NP
  - **Interactive proofs**
- Zero Knowledge Proofs
  - **ZK proofs, intuitive**
  - **ZK proofs**
  - ZK proofs for NP problems
  - Commitment scemes

Overview

In this project:

Lectures:

- Complexity Theory
  - Turing Machines
  - Classes P and NP
  - Interactive proofs
- Zero Knowledge Proofs
  - ZK proofs, intuitive
  - **ZK proofs**
  - ZK proofs for NP problems
  - Commitment scemes

Overview

In this project:

Lectures:

- Complexity Theory
  - Turing Machines
  - Classes P and NP
  - Interactive proofs
- Zero Knowledge Proofs
  - ZK proofs, intuitive
  - ZK proofs
  - **ZK proofs for NP problems**
  - **Commitment scemes**

Overview

In this project:

Work for you:

- 3 exercise classes
  - Classes right after the lectures (with a 1 hour lunch break in between)
  - Hand in exercises the next day before 17:00 h
- 1 final assignment
  - Presentations on 22 January
  - You may work in pairs
- Grading: pass/fail

Historical context

- $\sim$ 1900 - 1930: formalising mathematics
  Hilbert: *Wir müssen wissen, wir werden wissen*
- 1931: Gödel's incompleteness theorems
- 1935 - 1936: Entscheidungsproblem solved by Church and Turing
  - Methods similar to Gödel
  - Needed to formalise 'computation' (Church - Turing Thesis)
  - Turing's solution: Turing machines.
- From this concept, modern computers were developed.

Alan Turing 1912 - 1954

- Founder of Computer Science and Artificial Intelligence
- Turing Test
- Cracked the Enigma Code
- Died a tragic death

A Theoretical Model for a Computer

stephengg, Flickr

A Theoretical Model for a Computer


stephengg, Flickr

Turing Machines are:

- Easy to understand
- As powerful as any other computer

A Theoretical Model for a Computer

stephengg, Flickr

Turing Machines are:

- Easy to understand
- As powerful as any other computer

Therefore they are used to:

- Find limits of computability
- Analyse algorithms: how long does a computation take?

Turing Machines

Turing Machines

	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	1

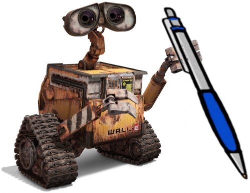
Turing Machines

	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	1

Turing Machines

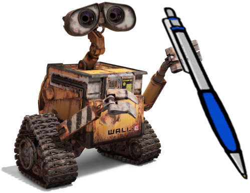
	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	1

Turing Machines



	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	1

Turing Machines



0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	0	1

Turing Machines

	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	0	1

Turing Machines

	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	0	1

Turing Machines

	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	0	1
	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

Turing Machines

	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	0	1
	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

Turing Machines

	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	0	1
	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Turing Machines

inputtape	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0
worktape	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
worktape	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
outputtape	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Input: x
Output: M(x)

Programming a Turing Machine

A Turing Machine needs instructions: write what? move where? These instructions depend on what the TM reads, and on its internal *state*. Notice: also need instruction: next state?

Suppose I want a TM changes the first two 1's it encounters to a 0, and then stops.

Programming a Turing Machine

A Turing Machine needs instructions: write what? move where? These instructions depend on what the TM reads, and on its internal *state*. Notice: also need instruction: next state?

Suppose I want a TM changes the first two 1's it encounters to a 0, and then stops.

Programming a Turing Machine

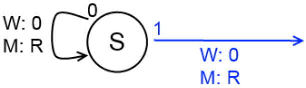
A Turing Machine needs instructions: write what? move where? These instructions depend on what the TM reads, and on its internal *state*. Notice: also need instruction: next state?

Suppose I want a TM changes the first two 1's it encounters to a 0, and then stops.

Programming a Turing Machine

A Turing Machine needs instructions: write what? move where? These instructions depend on what the TM reads, and on its internal *state*. Notice: also need instruction: next state?

Suppose I want a TM changes the first two 1's it encounters to a 0, and then stops.



Programming a Turing Machine

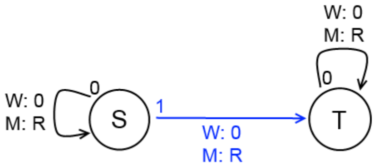
A Turing Machine needs instructions: write what? move where? These instructions depend on what the TM reads, and on its internal *state*. Notice: also need instruction: next state?

Suppose I want a TM changes the first two 1's it encounters to a 0, and then stops.

Programming a Turing Machine

A Turing Machine needs instructions: write what? move where? These instructions depend on what the TM reads, and on its internal *state*. Notice: also need instruction: next state?

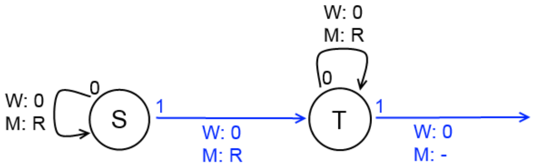
Suppose I want a TM changes the first two 1's it encounters to a 0, and then stops.



Programming a Turing Machine

A Turing Machine needs instructions: write what? move where? These instructions depend on what the TM reads, and on its internal *state*. Notice: also need instruction: next state?

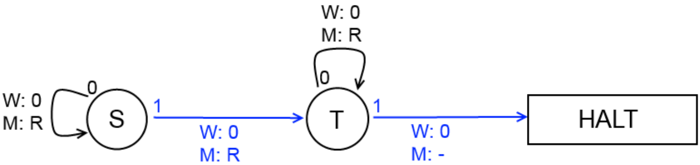
Suppose I want a TM changes the first two 1's it encounters to a 0, and then stops.



Programming a Turing Machine

A Turing Machine needs instructions: write what? move where? These instructions depend on what the TM reads, and on its internal *state*. Notice: also need instruction: next state?

Suppose I want a TM changes the first two 1's it encounters to a 0, and then stops.



TM: formal mathematical definition

A Turing Machine is a tuple $< \Gamma, Q, \delta >$, consisting of:

TM: formal mathematical definition

A Turing Machine is a tuple $< \Gamma, Q, \delta >$, consisting of:

- Γ, the set of symbols allowed on the tape: the alphabet.

TM: formal mathematical definition

A Turing Machine is a tuple $< \Gamma, Q, \delta >$, consisting of:

- Γ, the set of symbols allowed on the tape: the alphabet.
- Q, the set of states, including an initial state S and a final state $HALT$

TM: formal mathematical definition

A Turing Machine is a tuple $< \Gamma, Q, \delta >$, consisting of:

- Γ, the set of symbols allowed on the tape: the alphabet.
- Q, the set of states, including an initial state S and a final state $HALT$
- $\delta : Q \times \Gamma \to \Gamma \times \{L, R, -\} \times Q$, the transition function, defining which symbol to write, where to move and which is the next state, depending on the current state and the symbol being read.

One step of a Turing Machine is one execution of the transition function.

TM: formal mathematical definition

A Turing Machine is a tuple $< \Gamma, Q, \delta >$, consisting of:

- Γ, the set of symbols allowed on the tape: the alphabet.
- Q, the set of states, including an initial state S and a final state $HALT$
- $\delta : Q \times \Gamma^k \to \Gamma^k \times \{L, R, -\}^k \times Q$, the transition function, defining which symbol to write, where to move and which is the next state, depending on the current state and the symbol being read.

One step of a Turing Machine is one execution of the transition function.

Turing Machines example: counting

Let $\Gamma = \{0, 1, blank\}$, $Q = \{S, 1, 2, halt\}$, and δ defined by:

State?	read?	write:	move:	next state:
state S	blank	blank	L	1
	0	0	R	0
	1	1	R	0
state 1	blank	1	R	2
	0	1	L	2
	1	0	L	1
state 2	blank	blank	R	halt
	0	0	L	2
	1	1	L	2

Movie of a working Turing Machine

Complexity classes

Catalog problems according to 'how hard they are to solve'.

Complexity classes

Catalog problems according to 'how hard they are to solve'.

Formalise 'solving a problem':

Complexity classes

Catalog problems according to 'how hard they are to solve'.

Formalise 'solving a problem':

Find an algorithm that decides for all relevant x whether $x \in L$
Note: usually, $L \subseteq \{0,1\}^*$

Complexity classes

Catalog problems according to 'how hard they are to solve'.

Formalise 'solving a problem':

Find an algorithm that decides for all relevant x whether $x \in L$
Note: usually, $L \subseteq \{0, 1\}^*$

Example: Which words are palindromes?

Complexity classes

Catalog problems according to 'how hard they are to solve'.

Formalise 'solving a problem':

Find an algorithm that decides for all relevant x whether $x \in L$
Note: usually, $L \subseteq \{0, 1\}^*$

Example: Which words are palindromes?
$\rightarrow \quad x \in \{\text{Palindromes}\}?$ (x = 'radar', 'mandarin', ...)

Complexity classes

Catalog problems according to 'how hard they are to solve'.

Formalise 'solving a problem':

Find an algorithm that decides for all relevant x whether $x \in L$
Note: usually, $L \subseteq \{0,1\}^*$

Example: Which words are palindromes?
$\rightarrow \quad x \in \{\text{Palindromes}\}$? (x = 'radar', 'mandarin', ...)

Find algorithm: build a TM M such that:

- $x \in L \iff M(x) = 1$
- $x \notin L \iff M(x) = 0$

Complexity classes

Catalog problems according to 'how hard they are to solve'.

Formalise 'hardness':

Complexity classes

Catalog problems according to 'how hard they are to solve'.

Formalise ‘hardness’:

Time/Space/Communication/... needed, as funtion of the input length.

Complexity classes

Catalog problems according to 'how hard they are to solve'.

Formalise 'hardness':

Time/Space/Communication/... needed, as funtion of the input length.

Time: number of steps TM takes to compute output.
Space: number of cells of working tape TM needs to compute output.

Complexity classes

Catalog problems according to 'how hard they are to solve'.

Formalise 'hardness':

Time/Space/Communication/... needed, as funtion of the input length.

Time: number of steps TM takes to compute output.
Space: number of cells of working tape TM needs to compute output.

Note: Complexity of a problem is independent of the model of TM (Exercise!)

P: Polynomial Time

Complexity Class P

All problems that are solved in polynomial time.

P: Polynomial Time

Complexity Class P

All problems that are solved in polynomial time.

Complexity Class P:
$L \in P$ if and only if:
$\exists c \in \mathbb{N}$ and a TM M such that for all x:

- M halts on input x within $|x|^c$ steps.

P: Polynomial Time

Complexity Class P

All problems that are solved in polynomial time.

Complexity Class P:
$L \in P$ if and only if:
$\exists c \in \mathbb{N}$ and a TM M such that for all x:

- M halts on input x within $|x|^c$ steps.
- $x \in L \iff M(x) = 1$
- $x \notin L \iff M(x) = 0$

P: example

Is x a palindrome?

NP: Nondeterministic Polynomial Time

Complexity Class NP

All problems *verifiable* in polynomial time.

NP: Nondeterministic Polynomial Time

Complexity Class NP

All problems *verifiable* in polynomial time.

Complexity Class NP:
$L \in$ NP if and only if:
$\exists c_1, c_2 \in \mathbb{N}$ and a TM M such that for all x there is a witness y such that:

NP: Nondeterministic Polynomial Time

Complexity Class NP

All problems *verifiable* in polynomial time.

Complexity Class NP:
$L \in$ NP if and only if:
$\exists c_1, c_2 \in \mathbb{N}$ and a TM M such that for all x there is a witness y such that:

- $|y| < |x|^{c_1}$

NP: Nondeterministic Polynomial Time

Complexity Class NP

All problems *verifiable* in polynomial time.

Complexity Class NP:
$L \in \mathrm{NP}$ if and only if:
$\exists c_1, c_2 \in \mathbb{N}$ and a TM M such that for all x there is a witness y such that:

- $|y| < |x|^{c_1}$
- M halts on input x, y within $|x|^{c_2}$ steps.

NP: Nondeterministic Polynomial Time

Complexity Class NP

All problems *verifiable* in polynomial time.

Complexity Class NP:
$L \in$ NP if and only if:
$\exists c_1, c_2 \in \mathbb{N}$ and a TM M such that for all x there is a witness y such that:

- $|y| < |x|^{c_1}$
- M halts on input x, y within $|x|^{c_2}$ steps.
- $x \in L \iff M(x, y) = 1$
- $x \notin L \iff M(x, y) = 0$

NP: example

SAT: Satisfiability of a boolean formula
Is there an assignment of True and False to the variables in a formula, such that the formula evaluates to True?

NP: example

SAT: Satisfiability of a boolean formula
Is there an assignment of True and False to the variables in a formula, such that the formula evaluates to True?

Input: a boolean formula ϕ in standard form

NP: example

SAT: Satisfiability of a boolean formula
Is there an assignment of True and False to the variables in a formula, such that the formula evaluates to True?

Input: a boolean formula ϕ in standard form
$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3)$

NP: example

SAT: Satisfiability of a boolean formula
Is there an assignment of True and False to the variables in a formula, such that the formula evaluates to True?

Input: a boolean formula ϕ in standard form

$(x_1 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_3)$

$(x_1 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_3) \land \neg x_1 \land (x_2 \lor \neg x_3) \land (x_1 \lor \neg x_2 \lor x_3)$

NP: example

SAT: Satisfiability of a boolean formula
Is there an assignment of True and False to the variables in a formula, such that the formula evaluates to True?

Input: a boolean formula ϕ in standard form

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3)$

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_1 \wedge (x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$

Time to compute solution: $\sim 2^{|\phi|}$

Exponential!

NP: example

SAT: Satisfiability of a boolean formula
Is there an assignment of True and False to the variables in a formula, such that the formula evaluates to True?

Input: a boolean formula ϕ in standard form
$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3)$
$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_1 \wedge (x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$
Time to compute solution: $\sim 2^{|\phi|}$
Exponential!

Witness: an assignment to the variables.

NP: example

SAT: Satisfiability of a boolean formula
Is there an assignment of True and False to the variables in a formula, such that the formula evaluates to True?

Input: a boolean formula ϕ in standard form
$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3)$
$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_1 \wedge (x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$
Time to compute solution: $\sim 2^{|\phi|}$
Exponential!

Witness: an assignment to the variables.
Time to check assignment: $\sim |\phi|$

P vs NP

Clearly $P \subseteq NP$
Open problem: $P = NP$?

General consensus: no.
We will use this later on!

[1] Knuth, Donald E. (May 20, 2014). "Twenty Questions for Donald Knuth". informit.com.

P vs NP

Clearly $P \subseteq NP$

Open problem: $P = NP$?

General consensus: no.

We will use this later on!

Fun Facts:

- Solving the P vs NP problem wins you a million dollars!

[1] Knuth, Donald E. (May 20, 2014). "Twenty Questions for Donald Knuth". informit.com.

P vs NP

Clearly $P \subseteq NP$
Open problem: $P = NP$?

General consensus: no.
We will use this later on!

Fun Facts:

- Solving the P vs NP problem wins you a million dollars!
- If P = NP ... could be a problem:
  - Digital security collapses
  - Mathematicians would be out of work

But, as Donald Knuth[1] reassures us: A proof of P = NP will almost certainly be non-constructive, so no worries there :)

[1] Knuth, Donald E. (May 20, 2014). "Twenty Questions for Donald Knuth". informit.com.