Lecture 2: Machine Models, Basic Computability Theory

Models of Computation

https://clegra.github.io/moc.html

Clemens Grabmayer

Ph.D. Program, Advanced Courses Period
Gran Sasso Science Institute
L'Aquila, Italy

July 7, 2026

Course overview

Monday, July 6 10.30 – 12.30	Tuesday, July 7 10.30 – 12.30	Wednesday, July 8 10.30 – 12.30	Thursday, July 9 10.30 – 12.30	Friday, July 10
intro	*classic models*			*additional models*
Introduction to Computability	**Machine Models**	**Recursive Functions**	**Lambda Calculus**	
computation and decision problems, from logic to computability, overview of models of computation relevance of MoCs	Post Machines, typical features, Turing's analysis of human computers, Turing machines, basic recursion theory	primitive recursive functions, Gödel–Herbrand recursive functions, partial recursive funct's, partial recursive = = Turing-computable, Church's Thesis	λ-terms, β-reduction, λ-definable functions, partial recursive = λ-definable = Turing computable	
	imperative programming	*algebraic programming*	*functional programming*	
				14.30 – 16.30
				Three more Models of Computation
				Post's Correspondence Problem, Interaction-Nets, Fractran
				comparing computational power

Overview

- Post machine
- Turing machine
  - Turing's analysis of computations done by (human) computers

Overview

- Post machine
- Turing machine
  - Turing's analysis of computations done by (human) computers
  - formal definition
  - video
  - typical features of models of computation
  - Busy Beavers

Overview

- Post machine
- Turing machine
  - Turing's analysis of computations done by (human) computers
  - formal definition
  - video
  - typical features of models of computation
  - Busy Beavers
- Elementary recursion theory
  - an unsolvable problem
  - Halting problem
  - recursively enumerable, and recursive sets
  - universal language
  - Chomsky hierarchy

(Comput.) Yes-or-no-questions / Decision problems

Example

Tautology Problem for the propositional calculus
Instance: A formula ϕ of propositional logic.
Question: Is ϕ a tautology?

Suppose $A \subseteq E$, where E a set of finitely describable objects.

A decision method for A in E is a method by which, given an element $a \in E$, we can decide in a finite number of steps whether or not $a \in A$.

Decision problem for A in E: Find a decision method for A in E, or show that no such method can exist.

The decision problem for A in E is solvable (the set A in E is (effectively) calculable) if there exists a decision method for A in E.

(Comput.) What-questions / Computation Problems

Example

Computing the greatest common divisor
Instance: a pair $\langle a, b \rangle$ of numbers $a, b \in \mathbb{N}$ with $a, b > 0$.
Question: What is $\gcd(a, b)$, the greatest common divisor of a and b?

Suppose $F : A \to B$ is a mapping, where the elements of A, B are finitely describable objects.

A computation method for F is a method by which, given an element $a \in A$, we can obtain solution $F(a)$ in a finite number of steps.

Computation problem for F: Find a computation method for F, or show that no such method can exist.

A mapping F is calculable if there exists a computation method for F.

Representing function

Let $P(a_1, \ldots, a_n)$ be an n-ary number-theoretic predicate.

The representing function f of P:

$$f(a_1, \ldots, a_n) := \begin{cases} 1 & \ldots P(a_1, \ldots, a_n) \text{ is true} \\ 0 & \ldots P(a_1, \ldots, a_n) \text{ is false} \end{cases}$$

Hence:
A decision procedure can be handled as a computation procedure f by taking ‘0’ for ‘yes’, and ‘1’ for ‘no’.

Decision / Computation procedures (steps)

What is a computation method (procedure) more precisely, with respect to its steps?

– A mechanical, algorithmic computation procedure that:

- can be carried out by a machine M (ideal, not limited by resource problems, mechanical breakdown, etc.).
- for computing a function F on an argument a,
  - a is placed on the input device of the M,
  - which then produces $F(a)$ after finitely many steps.
- for computing a function F,
  - the machine M that is chosen for obtaining $F(a)$ may not be different for different arguments a

– Similar for a decision methods.

Solvability by an effective procedure

Q: Is the diophantine equation $15x + 9y + 12 = 0$ solvable?
(I.e. solvable for $x, y \in \mathbb{Z}$?)

From elementary number theory we know:

$$ax + by + c = 0 \text{ solvable in } \mathbb{Z} \iff \gcd(a, b) \mid c \qquad (\star)$$

Solvability by an effective procedure

Q: Is the diophantine equation $15x + 9y + 12 = 0$ solvable?
(I.e. solvable for $x, y \in \mathbb{Z}$?)

From elementary number theory we know:

$$ax + by + c = 0 \text{ solvable in } \mathbb{Z} \iff \gcd(a, b) \mid c \qquad (\star)$$

Using Euclid's algorithm we calculate $\gcd(15, 9)$:

15	:	9	=	1	rem	6
9	:	6	=	1	rem	3
6	:	3	=	2	rem	0

We find: $\gcd(15, 9) = 3$.
Due to $3 \mid 12$ and (**??**) we conclude:

A: Yes. (Infinitely many solutions, e.g. $x = 4$ and $y = -8$.)

Not effectively calculable

Examples (Shoenfield)

- methods that involve chance procedures: tossing a coin

Not effectively calculable

Examples (Shoenfield)

- methods that involve chance procedures: tossing a coin
- methods involving magic: asking a fortune teller

Not effectively calculable

Examples (Shoenfield)

- methods that involve chance procedures: tossing a coin
- methods involving magic: asking a fortune teller
- methods that require (unformalised, unmechanised) insight

Effectively calculable?

Example

Hilbert's 10th Problem

Instance: An equation $p(x_1, \ldots, x_n) = 0$, where
p a polynomial with integer coefficients.

Question: Is the equation solvable for $x_1, \ldots, x_n \in \mathbb{Z}$?

Instances based on quadratic polynomials are of the form $ax^2 + bxy + cy^2 + dx + ey + f = 0$ with $a, b, c, d, e, f \in \mathbb{Z}$.

Effectively calculable? – No!

Example

Hilbert's 10th Problem

Instance: An equation $p(x_1, \ldots, x_n) = 0$, where
p a polynomial with integer coefficients.

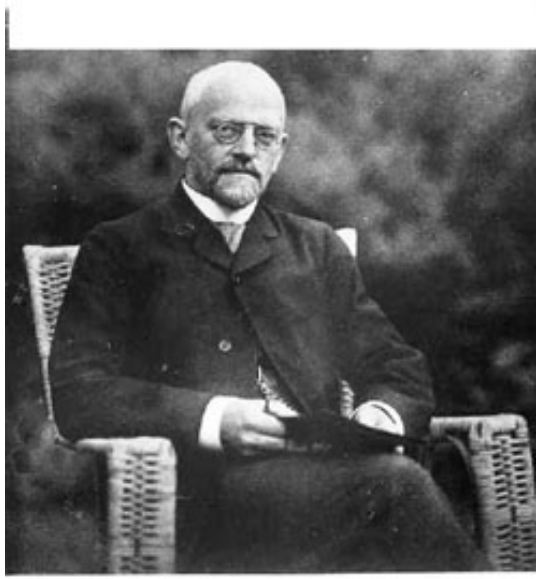
Question: Is the equation solvable for $x_1, \ldots, x_n \in \mathbb{Z}$?

Instances based on quadratic polynomials are of the form $ax^2 + bxy + cy^2 + dx + ey + f = 0$ with $a, b, c, d, e, f \in \mathbb{Z}$.

Theorem (Matijasevic, 1970)

Hilbert's 10th Problem is unsolvable.

David Hilbert (1862–1943)



Problem (Entscheidungsproblem, 1928)

Is there a method for deciding, given a formula ϕ of the predicate calculus, whether or not ϕ is a tautology?

Timeline: From logic to computability

1900	Hilbert's 23 Problems in mathematics
1921	Schönfinkel: Combinatory logic
1928	Hilbert/Ackermann: formulate completeness/decision problems for the predicate calculus (the latter called 'Entscheidungsproblem')
1929	Presburger: completeness/decidability of theory of addition on $\mathbb{Z}$
1930	Gödel: completeness theorem of predicate calculus
1931	Gödel: incompleteness theorems for first-order arithmetic
1932	Church: λ-calculus
1933/34	Herbrand/Gödel: general recursive functions
1936	Church/Kleene: λ-definable $\sim$ general recursive Church Thesis: 'effectively calculable' be defined as either Church shows: the 'Entscheidungsproblem' is unsolvable
1937	Post: machine model; Church's thesis as 'working hypothesis' Turing: convincing analysis of a 'human computer' leading to the 'Turing machine'

Reading recommended

1. Post machine: Page 1 + first paragraph on page 2 of:
   - Emil Post: *Finite Combinatory Processes – Formulation 1*, Journal of Symbolic Logic (1936), [4].

2. Turing machine motivation:
   Turing's analysis of a human computer:
   Part I of Section 9, pp. 249–252 of:
   - Alan M. Turing's: *On computable numbers, with an application to the Entscheidungsproblem*', Proceedings of the London Mathematical Society (1936), [5].

Emil Post

Emil Leon Post (1897–1954)

Post about . . .

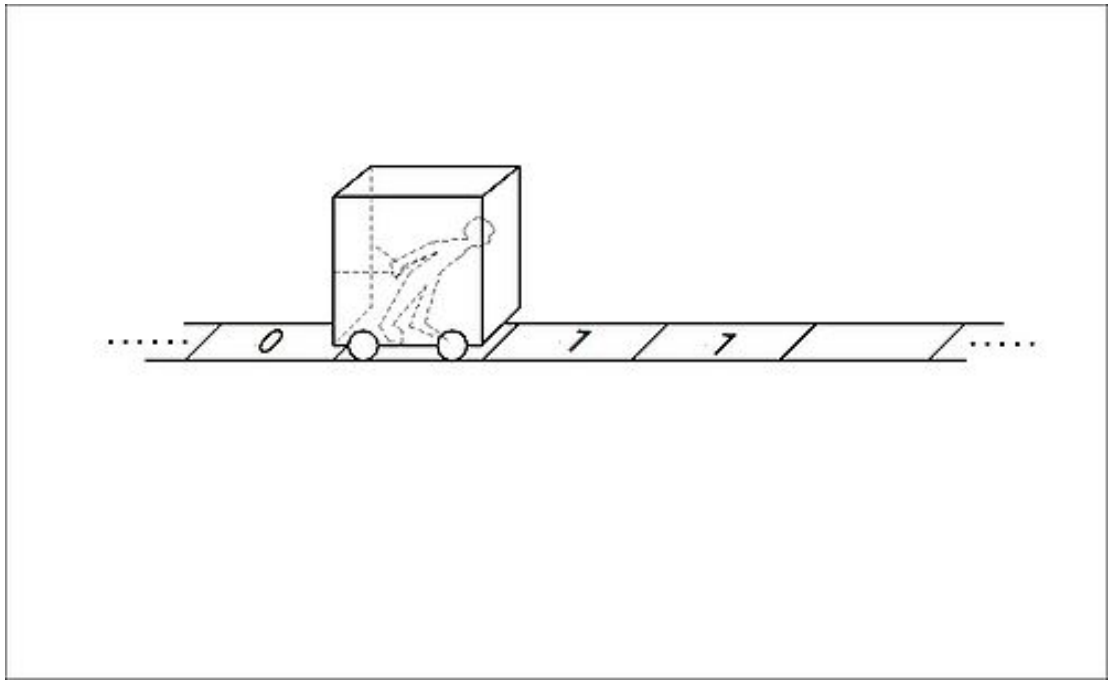
. . . a result of his from 1921 similar to the Incompleteness Theorem:

Theorem (Gödel, 1931 (paraphrased here))

Every axiomatisable, consistent first-order-logic system of number theory is incomplete: it contains true, but unprovable formulas.

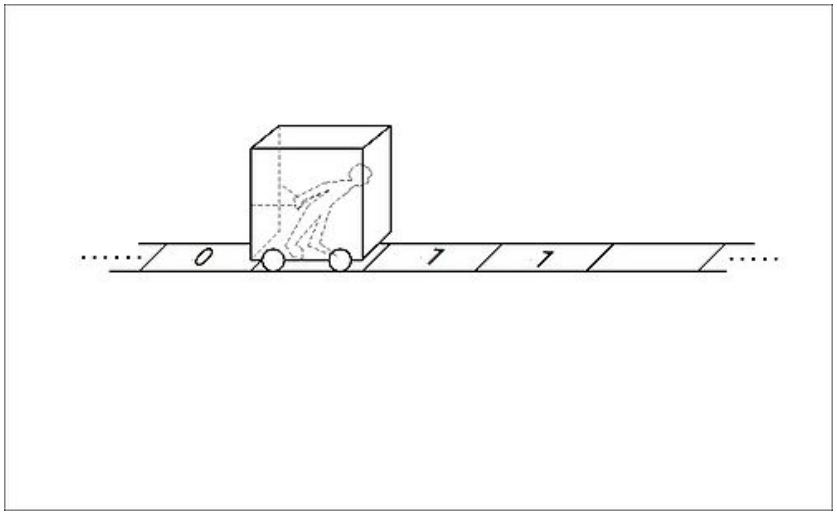
> *"For full generality a complete analysis would have to be given of all possible ways in which the human mind could set up finite processes for generating sequences."*

Post machine (1936)



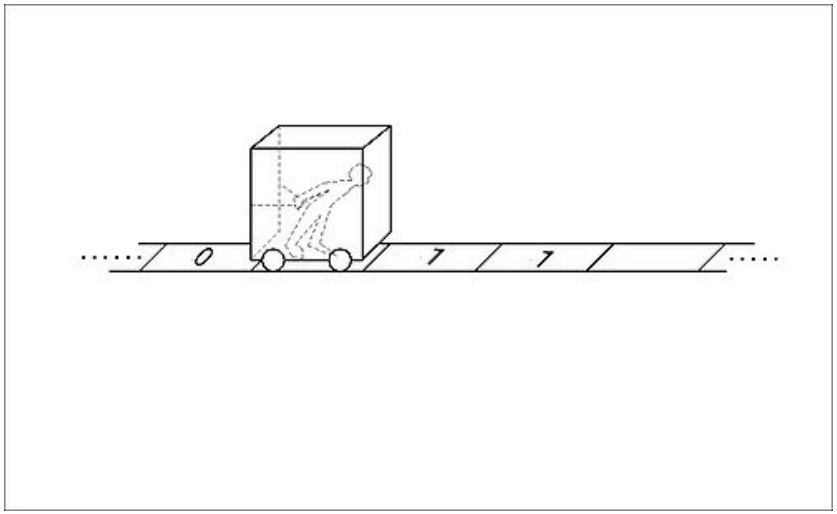
Emil Post: *Finite Combinatory Processes – Formulation 1* (1936), Journal of Symbolic Logic, [4].

Post machine (1936)



“The worker is assumed to be capable of performing the following primitive acts:

Post machine (1936)



"The worker is assumed to be capable of performing the following primitive acts:

(a) Marking the box he is in (assumed empty),

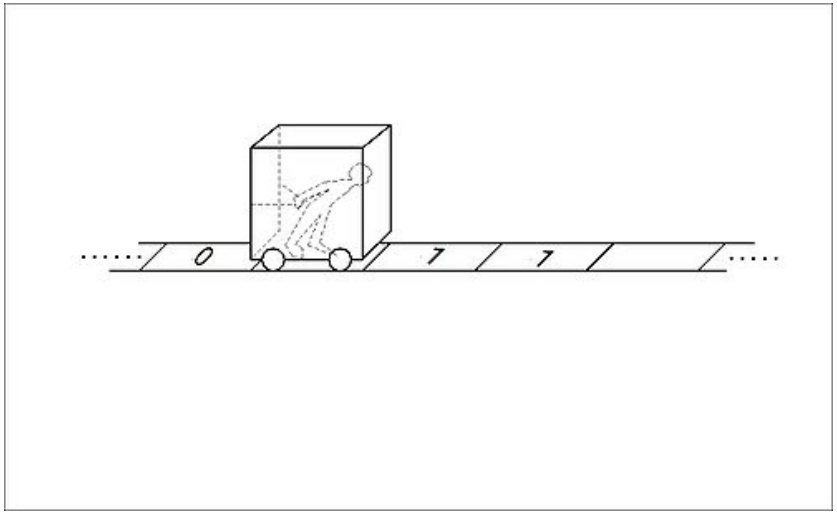
(b) Erasing the mark in the box he is in (assumed marked),

(c) Moving to the box on his right,

(d) Moving to the box on his left,

(e) Determining whether the box he is in, is or is not marked."

Post machine (1936)



‘Directions’ (= list of instructions):

- Start at the starting point and follow direction 1.
- Then a finite number of directions numbered 1, 2, 3, . . . , n, where the i-th has one of the following forms:
  - Perform operation $O_i \in \{(a), (b), (c), (d)\}$, then follow direction j_i.
  - Perform operation (e) and according as the answer is yes or no correspondingly follow direction j_i' or j_i''.
  - Stop.

Exercise

Exercise

Construct a Post machine that adds one to a natural number in unary representation.

Typical features of 'computationally complete' MoC's

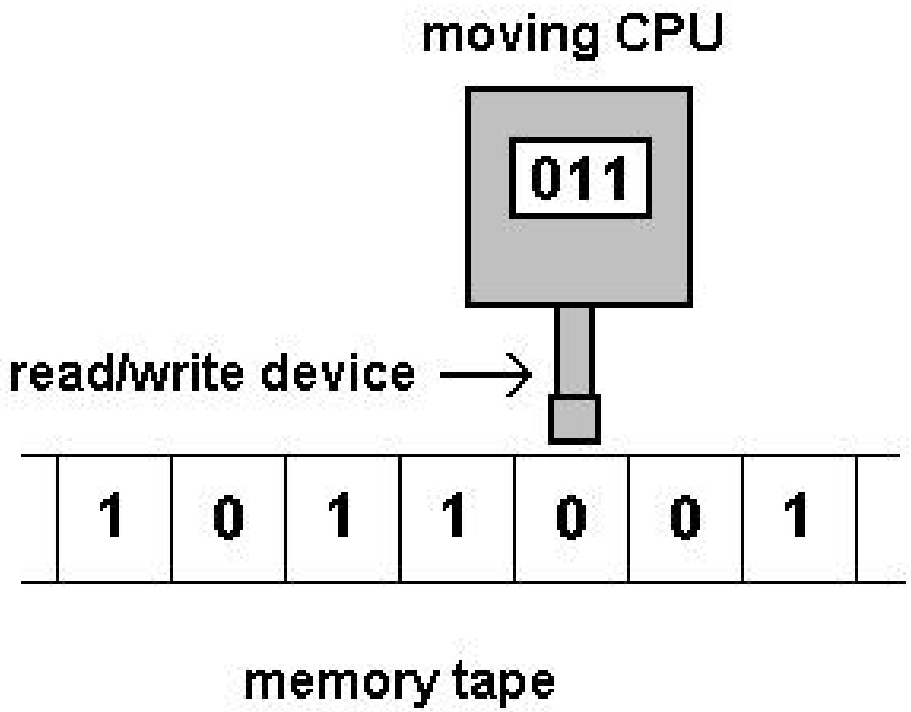
- storage (unbounded)
- control (finite, given)
- modification
  - of (immediately accessible) stored data
  - of control state
- conditionals
- loop (unbounded)
- stopping condition

(Credits due to: Vincent van Oostrom)

Turing computability

Alan Turing (1912 –1954)

Turing machine



Turing machine: formal definition

Definition

A Turing machine is a tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \textvisiblespace, F \rangle$ where:

Turing machine: formal definition

Definition

A Turing machine is a tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \textvisiblespace, F \rangle$ where:

- Q is a finite set of states;
- $q_0 \in Q$ is called the initial state;
- $F \subseteq Q$ is the set of final or accepting states.

Turing machine: formal definition

Definition

A Turing machine is a tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \textbb, F \rangle$ where:

- Q is a finite set of states;
- Σ is the input alphabet;
- Γ is the tape alphabet that is finite and $\Gamma \supseteq \Sigma \cup \{\textbb\}$ holds;
- $\textbb$ is a designated blank symbol not contained in Σ;
- $q_0 \in Q$ is called the initial state;
- $F \subseteq Q$ is the set of final or accepting states.

Turing machine: formal definition

Definition

A Turing machine is a tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \textbb, F \rangle$ where:

- Q is a finite set of states;
- Σ is the input alphabet;
- Γ is the tape alphabet that is finite and $\Gamma \supseteq \Sigma \cup \{\textbb\}$ holds;
- $\delta : (Q \setminus F) \times \Gamma \rightharpoonup Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is a partial function, called the transition function;
- $\textbb$ is a designated blank symbol not contained in Σ;
- $q_0 \in Q$ is called the initial state;
- $F \subseteq Q$ is the set of final or accepting states.

Turing's analysis of a human 'computer'

Section 9 in Turing's 1937 paper 'On computable numbers, with an application to the Entscheidungsproblem' [5].

A direct appeal to intuition in analysing human computation:

- paper is divided into squares
- one-dimensional paper ('tape' divided into squares)
- number of symbols is finite
- behaviour of computer at any time is determined by:
  - observed symbols
  - her/his 'state of mind'
- bound B on the number of symbols/squares
  that the computer can observe at any moment
- number of 'states of mind' of the computer is finite

Turing's analysis of a human 'computer'

- modification of tape symbols

Turing's analysis of a human 'computer'

- modification of tape symbols
  - in a simple operation only one symbol is altered
  - only 'observed' symbols can be altered

Turing's analysis of a human 'computer'

- modification of tape symbols
  - in a simple operation only one symbol is altered
  - only 'observed' symbols can be altered
- modification of observed squares

Turing's analysis of a human 'computer'

- modification of tape symbols
  - in a simple operation only one symbol is altered
  - only 'observed' symbols can be altered
- modification of observed squares
  - new observed squares are within L squares of a previously observed square
  - other directly observable squares? – T. argues: not necessary

Turing's analysis of a human 'computer'

- modification of tape symbols
  - in a simple operation only one symbol is altered
  - only 'observed' symbols can be altered
- modification of observed squares
  - new observed squares are within L squares of a previously observed square
  - other directly observable squares? – T. argues: not necessary
- modification of 'state of mind'

Turing's analysis of a human 'computer'

- simple operations must include:

Turing's analysis of a human 'computer'

- simple operations must include:
  - (a) change of a single symbol on one of the B observed squares
  - (b) change of one of the observed squares to another square at most L squares away

Turing's analysis of a human 'computer'

- simple operations must include:
  - (a) change of a single symbol on one of the B observed squares
  - (b) change of one of the observed squares to another square
    at most L squares away
- most general simple operations:

Turing's analysis of a human 'computer'

- simple operations must include:
  - (a) change of a single symbol on one of the B observed squares
  - (b) change of one of the observed squares to another square
    at most L squares away
- most general simple operations:
  - A change (a) of symbol with a possible change of state of mind
  - A change (b) of observed square,
    together with a possible change of state of mind.

Turing's analysis of a human 'computer'

- simple operations must include:
  - (a) change of a single symbol on one of the B observed squares
  - (b) change of one of the observed squares to another square
    at most L squares away
- most general simple operations:
  - A change (a) of symbol with a possible change of state of mind
  - A change (b) of observed square,
    together with a possible change of state of mind.

"The machines just described do not differ very essentially from [... Turing machines ...], and to any machine of this type a computing machine can be constructed to compute the same sequence, that is to say the sequence computed by the computer."

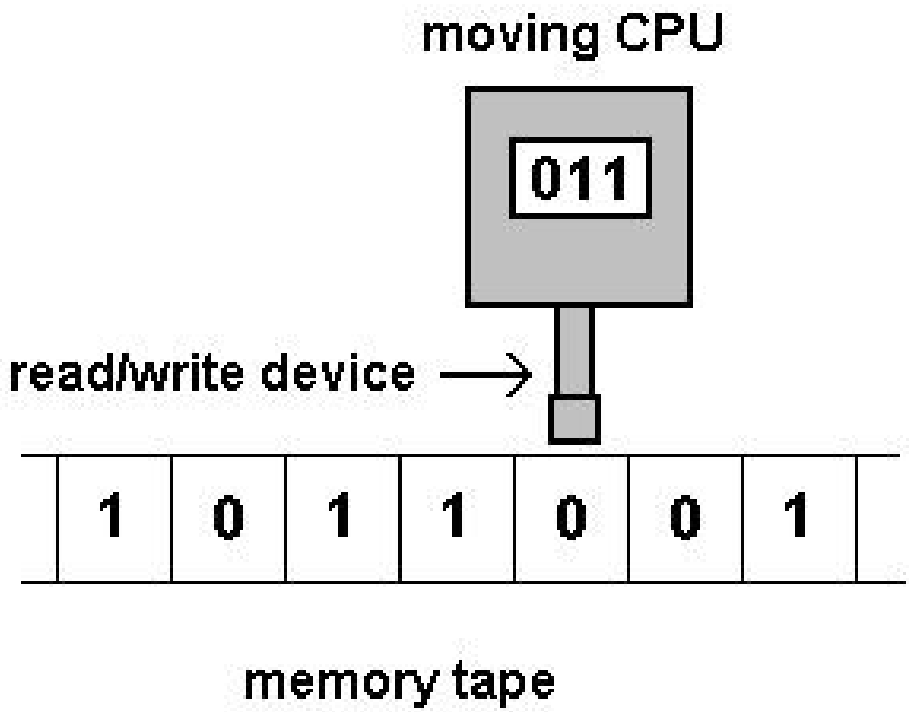
Turing's analysis of a human 'computer'

- simple operations must include:
  - (a) change of a single symbol on one of the B observed squares
  - (b) change of one of the observed squares to another square at most L squares away
- most general simple operations:
  - A change (a) of symbol with a possible change of state of mind
  - A change (b) of observed square, together with a possible change of state of mind.

"The machines just described do not differ very essentially from [... Turing machines ...], and to any machine of this type a computing machine can be constructed to compute the same sequence, that is to say the sequence computed by the computer."

"It is my contention that these operations include all those which are used in the computation of a number."

Turing machine



Church–Turing Thesis

Thesis (Church–Turing, 1937)

Every effectively calculable function is computable by a Turing-machine.

Turing machine: formal definition

Definition

A Turing machine is a tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \textbb, F \rangle$ where:

- Q is a finite set of states;
- Σ is the input alphabet;
- Γ is the tape alphabet that is finite and $\Gamma \supseteq \Sigma \cup \{\textbb\}$ holds;
- $\delta : (Q \setminus F) \times \Gamma \rightharpoonup Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is a partial function, called the transition function;
- $\textbb$ is a designated blank symbol not contained in Σ;
- $q_0 \in Q$ is called the initial state;
- $F \subseteq Q$ is the set of final or accepting states.

Turing machine: definition notions

Definition

Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \textbb, F \rangle$ be a Turing machine.

A configuration of M is an element $w_1 q w_2 \in \Gamma^* \times Q \times \Gamma^*$ such that the first letter in w_1 and the last letter in w_2 are different from $\textbb$.

Turing machine: definition notions

Definition

Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not b, F \rangle$ be a Turing machine.

A configuration of M is an element $w_1 q w_2 \in \Gamma^* \times Q \times \Gamma^*$ such that the first letter in w_1 and the last letter in w_2 are different from $\not b$.

- $uqav'$ with $a \in \Sigma$ is an end-configuration if $\delta(q, a)$ is undefined.

Turing machine: definition notions

Definition

Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not b, F \rangle$ be a Turing machine.

A configuration of M is an element $w_1 q w_2 \in \Gamma^* \times Q \times \Gamma^*$ such that the first letter in w_1 and the last letter in w_2 are different from $\not b$.

- $uqav'$ with $a \in \Sigma$ is an end-configuration if $\delta(q, a)$ is undefined.
- uqv' is accepting configuration if $q \in F$.

Turing machine: definition notions

Definition

Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not b, F \rangle$ be a Turing machine.

A configuration of M is an element $w_1 q w_2 \in \Gamma^* \times Q \times \Gamma^*$ such that the first letter in w_1 and the last letter in w_2 are different from $\not b$.

- $uqav'$ with $a \in \Sigma$ is an end-configuration if $\delta(q, a)$ is undefined.
- uqv' is accepting configuration if $q \in F$.

$\vdash_M$ ... next-move-relation
$\vdash_M^*$ ... reflexive, and transitive closure of $\vdash_M$

Turing machine: definition notions

Definition

Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not b, F \rangle$ be a Turing machine.

A configuration of M is an element $w_1 q w_2 \in \Gamma^* \times Q \times \Gamma^*$ such that the first letter in w_1 and the last letter in w_2 are different from $\not b$.

- $uqav'$ with $a \in \Sigma$ is an end-configuration if $\delta(q, a)$ is undefined.
- uqv' is accepting configuration if $q \in F$.

$\vdash_M$ ... next-move-relation
$\vdash_M^*$ ... reflexive, and transitive closure of $\vdash_M$

Let $w \in \Sigma^*$.

- M halts on (input) w if $q_0 w \vdash_M^* uqv$ for some end-config. uqv.

Turing machine: definition notions

Definition

Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not b, F \rangle$ be a Turing machine.

A configuration of M is an element $w_1 q w_2 \in \Gamma^* \times Q \times \Gamma^*$ such that the first letter in w_1 and the last letter in w_2 are different from $\not b$.

- $uqav'$ with $a \in \Sigma$ is an end-configuration if $\delta(q, a)$ is undefined.
- uqv' is accepting configuration if $q \in F$.

$\vdash_M$ ... next-move-relation
$\vdash_M^*$ ... reflexive, and transitive closure of $\vdash_M$

Let $w \in \Sigma^*$.

- M halts on (input) w if $q_0 w \vdash_M^* uqv$ for some end-config. uqv.
- M accepts w if $q_0 w \vdash_M^* uqv$ for some accepting config. uqv.

Turing machine: definition notions

Definition

Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \textvisiblespace, F \rangle$ be a Turing machine.

A configuration of M is an element $w_1 q w_2 \in \Gamma^* \times Q \times \Gamma^*$ such that the first letter in w_1 and the last letter in w_2 are different from $\textvisiblespace$.

- $uqav'$ with $a \in \Sigma$ is an end-configuration if $\delta(q, a)$ is undefined.
- uqv' is accepting configuration if $q \in F$.

$\vdash_M$ ... next-move-relation
$\vdash_M^*$ ... reflexive, and transitive closure of $\vdash_M$

Let $w \in \Sigma^*$.

- M halts on (input) w if $q_0 w \vdash_M^* uqv$ for some end-config. uqv.
- M accepts w if $q_0 w \vdash_M^* uqv$ for some accepting config. uqv.

$L(M) := \{w \in \Sigma^* \mid M \text{ accepts } w\}$ is the language accepted by M.

Recursively enumerable, and recursive languages

Definition

Let $L \subseteq \Sigma^*$ be a language.

L is called recursively enumerable if

- $L = L(M)$ for some Turing machine M with input symbols Σ.

L is called recursive if

- there is a Turing machine M with input symbols Σ such that
  - $L = L(M)$
  - M halts on all of its inputs.

Turing-computable (total) functions

Definition

A total function $f : \mathbb{N}^k \to \mathbb{N}$ is Turing-computable if there exists a Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \text{␣}, F \rangle$ and a calculable coding function $\langle \cdot \rangle : \mathbb{N} \to \Sigma^*$ such that:

- for all $n_1, \ldots, n_k \in \mathbb{N}$ there exists $q \in F$ such that:

$$q_0 \langle n_1 \rangle \text{␣} \langle n_2 \rangle \text{␣} \ldots \text{␣} \langle n_k \rangle \vdash_M^* q \langle f(n_1, \ldots, n_k) \rangle$$

Exercises

Exercise

Construct a Turing machine that adds one to a natural number in binary representation.

(In the film this Turing machine is executed five times consecutively.)

Exercises

Exercise

Construct a Turing machine that adds one to a natural number in binary representation.

(In the film this Turing machine is executed five times consecutively.)

Exercise

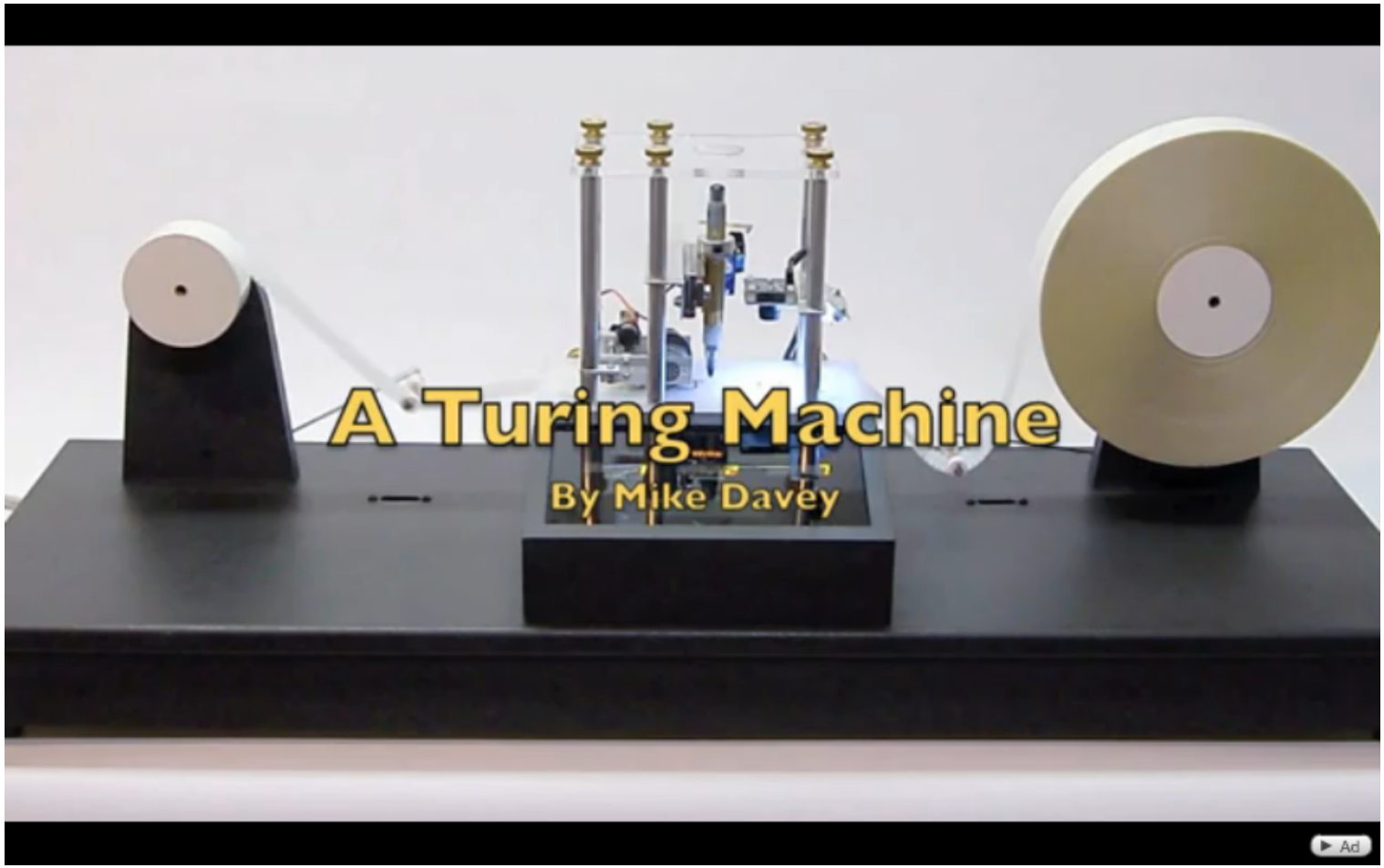
Construct a Turing machine that, if started on the empty tape, writes the sequence

$$0101101110111101111 0 \ldots$$

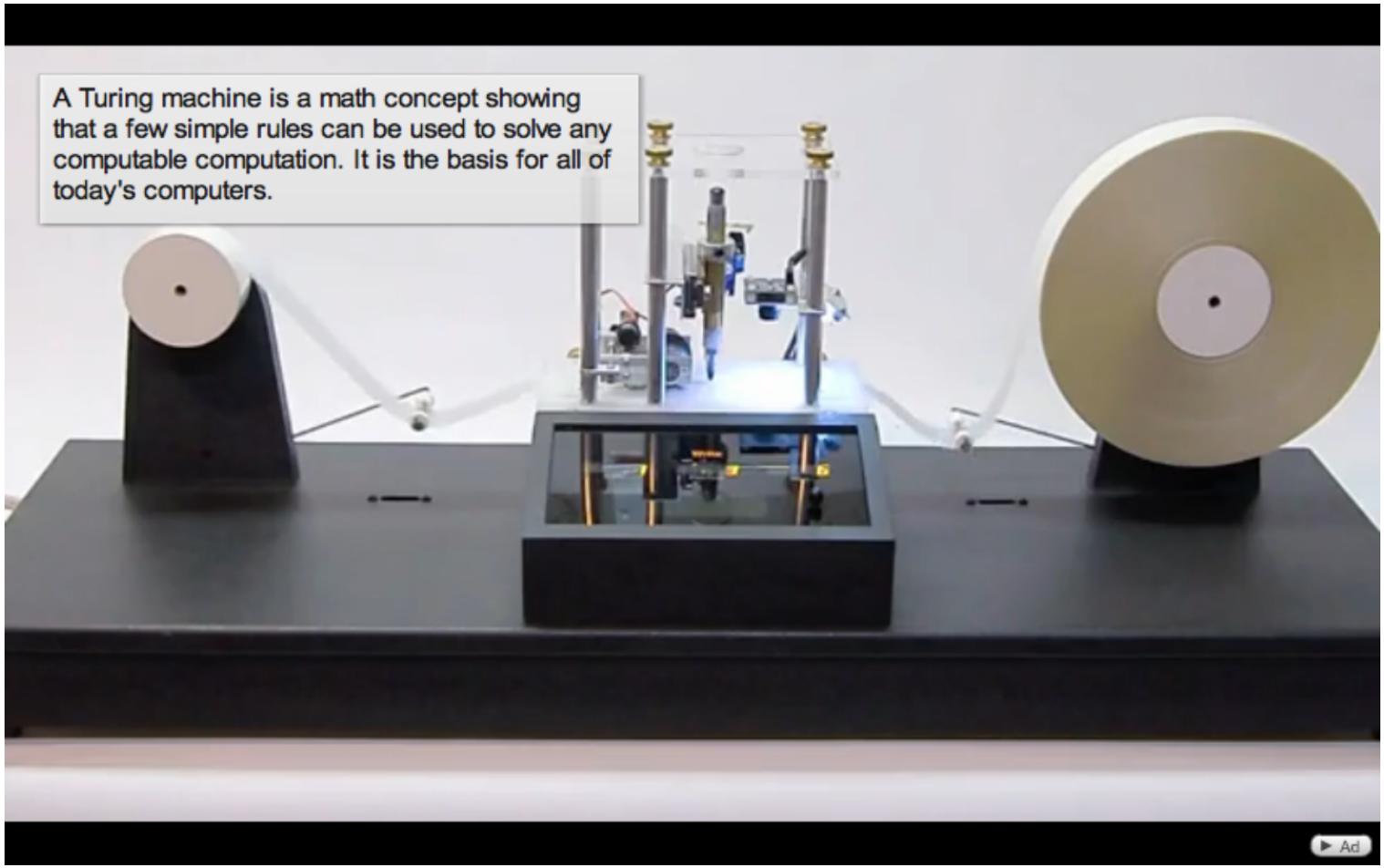
on the tape, but does not halt.

(Compare your machine with Turing's machine for this purpose.)

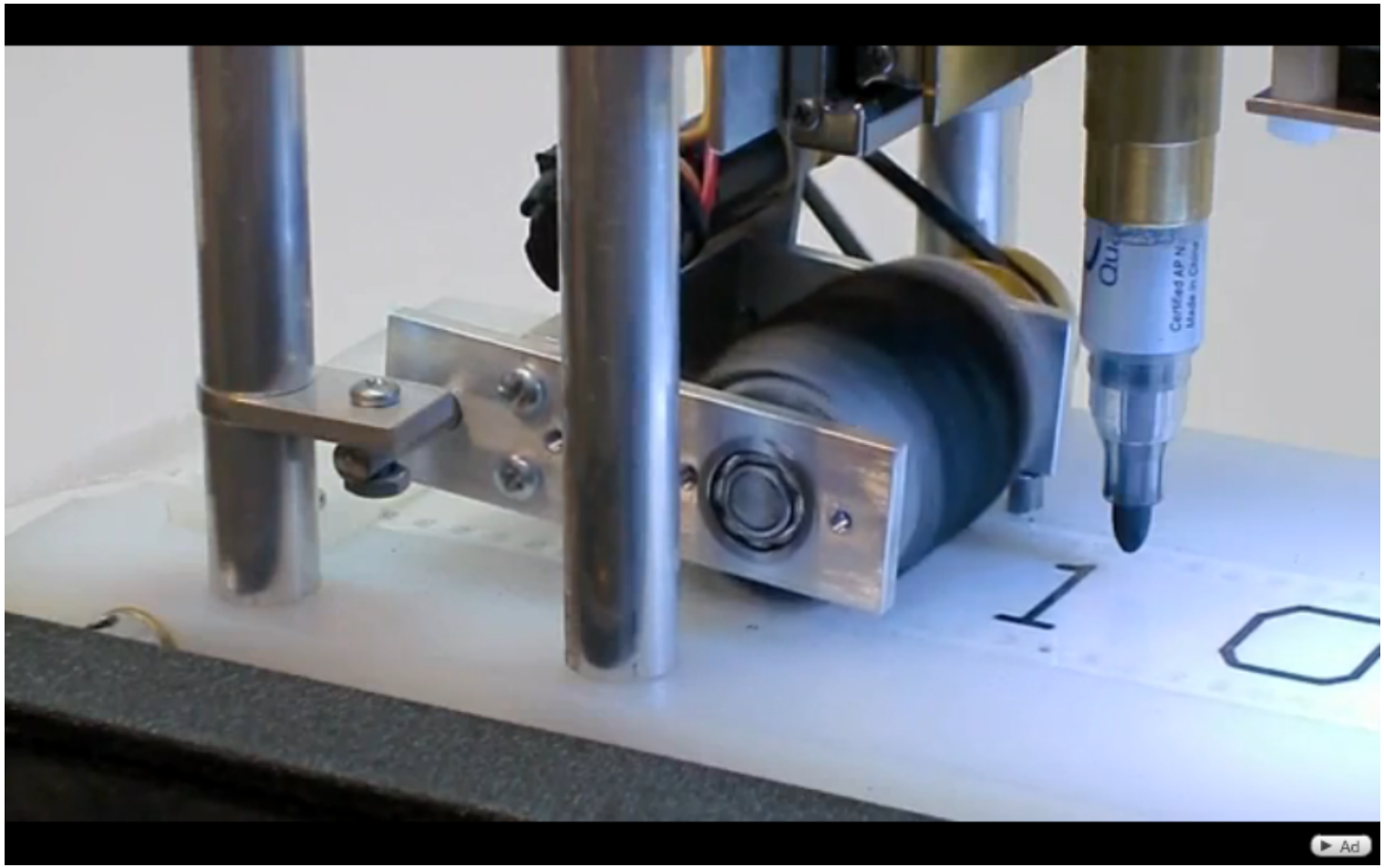
Mike Davey's Turing machine (link)



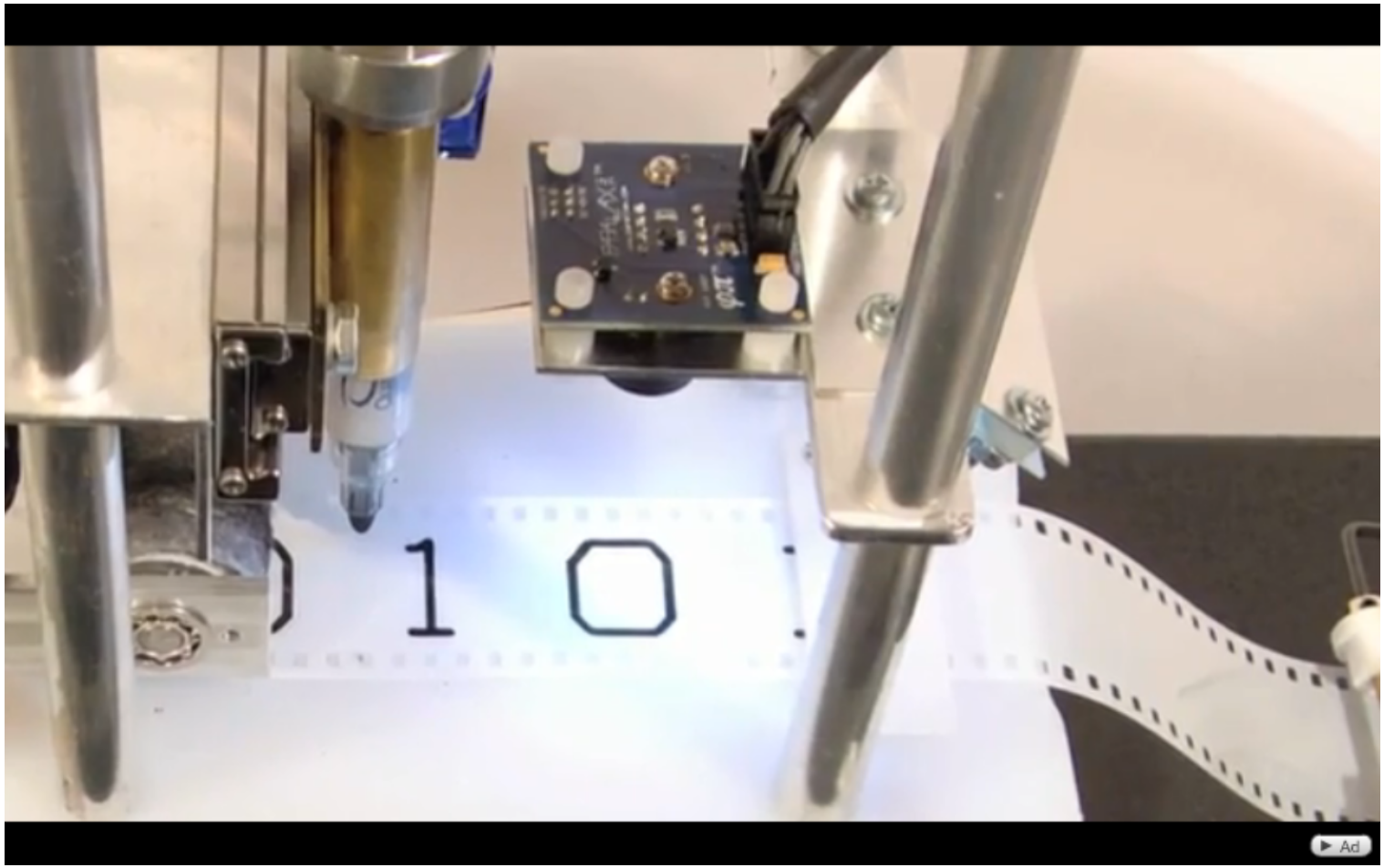
Mike Davey's Turing machine (link)



Mike Davey's Turing machine (link)



Mike Davey's Turing machine (link)



Mike Davey's Turing machine (link)

Typical features of ‘computationally complete’ MoC’s

- storage (unbounded)
- control (finite, given)
- modification
  - of (immediately accessible) stored data
  - of control state
- conditionals
- loop (unbounded)
- stopping condition

Variants of Turing machines

- TM's with semi-infinite tapes (infinite in only one direction)

Variants of Turing machines

- TM's with semi-infinite tapes (infinite in only one direction)
- TM's with multiple tapes

Variants of Turing machines

- TM's with semi-infinite tapes (infinite in only one direction)
- TM's with multiple tapes
  - Input/Output Turing machines (with input- and output tapes)

Variants of Turing machines

- TM's with semi-infinite tapes (infinite in only one direction)
- TM's with multiple tapes
  - Input/Output Turing machines (with input- and output tapes)
- non-deterministic TM's: $\delta \subseteq ((Q \times \Gamma) \times (Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}))$

Variants of Turing machines

- TM's with semi-infinite tapes (infinite in only one direction)
- TM's with multiple tapes
  - Input/Output Turing machines (with input- and output tapes)
- non-deterministic TM's: $\delta \subseteq ((Q \times \Gamma) \times (Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}))$
- tape-bounded TM's (by $f(n)$ for inputs of length n)

Variants of Turing machines

- TM's with semi-infinite tapes (infinite in only one direction)
- TM's with multiple tapes
  - Input/Output Turing machines (with input- and output tapes)
- non-deterministic TM's: $\delta \subseteq ((Q \times \Gamma) \times (Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}))$
- tape-bounded TM's (by $f(n)$ for inputs of length n)
- oracle Turing machines

Variants of Turing machines

- TM's with semi-infinite tapes (infinite in only one direction)
- TM's with multiple tapes
  - Input/Output Turing machines (with input- and output tapes)
- non-deterministic TM's: $\delta \subseteq ((Q \times \Gamma) \times (Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}))$
- tape-bounded TM's (by $f(n)$ for inputs of length n)
- oracle Turing machines
- Turing machines with advice

Variants of Turing machines

- TM's with semi-infinite tapes (infinite in only one direction)
- TM's with multiple tapes
  - Input/Output Turing machines (with input- and output tapes)
- non-deterministic TM's: $\delta \subseteq ((Q \times \Gamma) \times (Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}))$
- tape-bounded TM's (by $f(n)$ for inputs of length n)
- oracle Turing machines
- Turing machines with advice
- alternating Turing machines

Variants of Turing machines

- TM's with semi-infinite tapes (infinite in only one direction)
- TM's with multiple tapes
  - Input/Output Turing machines (with input- and output tapes)
- non-deterministic TM's: $\delta \subseteq ((Q \times \Gamma) \times (Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}))$
- tape-bounded TM's (by $f(n)$ for inputs of length n)
- oracle Turing machines
- Turing machines with advice
- alternating Turing machines
- ...

Variants of Turing machines

- TM's with semi-infinite tapes (infinite in only one direction)
- TM's with multiple tapes
  - Input/Output Turing machines (with input- and output tapes)
- non-deterministic TM's: $\delta \subseteq ((Q \times \Gamma) \times (Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}))$
- tape-bounded TM's (by $f(n)$ for inputs of length n)
- oracle Turing machines
- Turing machines with advice
- alternating Turing machines
- ...
- interactive/reactive TM's

Busy Beavers

Definition (Tibor Radó, 1962)

BB(n) := the largest number of steps any n-state Turing machine with tape alphabet $\{0, 1\}$ and blank symbol 0 can run before eventually halting, when started on an empty tape.

Busy Beavers

Definition (Tibor Radó, 1962)

$BB(n)$:= the largest number of steps any n-state Turing machine with tape alphabet $\{0, 1\}$ and blank symbol 0 can run before eventually halting, when started on an empty tape.

$BB(1) = 1$ (easy)

Busy Beavers

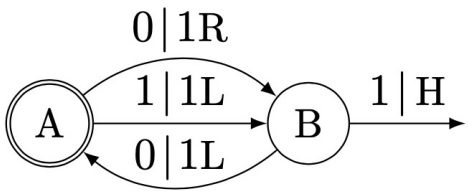
Definition (Tibor Radó, 1962)

$BB(n)$:= the largest number of steps any n-state Turing machine with tape alphabet $\{0, 1\}$ and blank symbol 0 can run before eventually halting, when started on an empty tape.

$BB(1) = 1$ (easy)

$BB(2) = 6$ (homework exercise)

2-State Busy Beaver



initial state	···	0	0	0	0A	0	0	···
step 1	···	0	0	0	1	0B	0	···
step 2	···	0	0	0	1A	1	0	···
step 3	···	0	0	0B	1	1	0	···
step 4	···	0	0A	1	1	1	0	···
step 5	···	0	1	1B	1	1	0	···
step 6 (halt)	···	0	1	1H	1	1	0	···

Figure 1: A 2-state Busy Beaver and its execution on an initially all-0 tape. Starting in state A, the machine finds a 0 on its tape and therefore follows the arrow labeled $0|1R$, which causes it to replace the 0 by a 1, move one square to the right, and transition into state B, and so on until the machine reaches the Halt arrow on its 6^{th} step.

Busy Beavers

Definition (Tibor Radó, 1962)

$BB(n)$:= the largest number of steps any n-state Turing machine with tape alphabet $\{0, 1\}$ and blank symbol 0 can run before eventually halting, when started on an empty tape.

$BB(1) = 1$	(easy)
$BB(2) = 6$	(homework exercise)
$BB(3) = 21$	(S. Lin and T. Radó, 1965)

Busy Beavers

Definition (Tibor Radó, 1962)

$BB(n)$:= the largest number of steps any n-state Turing machine with tape alphabet $\{0, 1\}$ and blank symbol 0 can run before eventually halting, when started on an empty tape.

$BB(1) = 1$	(easy)
$BB(2) = 6$	(homework exercise)
$BB(3) = 21$	(S. Lin and T. Radó, 1965)
$BB(4) = 107$	(A. Brady, 1983)

Busy Beavers

Definition (Tibor Radó, 1962)

$BB(n)$:= the largest number of steps any n-state Turing machine with tape alphabet $\{0, 1\}$ and blank symbol 0 can run before eventually halting, when started on an empty tape.

$BB(1) = 1$	(easy)
$BB(2) = 6$	(homework exercise)
$BB(3) = 21$	(S. Lin and T. Radó, 1965)
$BB(4) = 107$	(A. Brady, 1983)
$BB(5) = \begin{cases} \geq 47176870 \\ \\ \end{cases}$	(1990)

Busy Beavers

Definition (Tibor Radó, 1962)

$BB(n)$:= the largest number of steps any n-state Turing machine with tape alphabet $\{0, 1\}$ and blank symbol 0 can run before eventually halting, when started on an empty tape.

$BB(1) = 1$ (easy)

$BB(2) = 6$ (homework exercise)

$BB(3) = 21$ (S. Lin and T. Radó, 1965)

$BB(4) = 107$ (A. Brady, 1983)

$$BB(5) = \begin{cases} \geq 47176870 & \text{(1990)} \\ = 47176870 & \text{(international team, 2024)} \end{cases}$$

Busy Beavers

Definition (Tibor Radó, 1962)

$BB(n)$:= the largest number of steps any n-state Turing machine with tape alphabet $\{0, 1\}$ and blank symbol 0 can run before eventually halting, when started on an empty tape.

$BB(1) = 1$ (easy)

$BB(2) = 6$ (homework exercise)

$BB(3) = 21$ (S. Lin and T. Radó, 1965)

$BB(4) = 107$ (A. Brady, 1983)

$BB(5) = \begin{cases} \geq 47176870 \\ = 47176870 \end{cases}$ (1990) (international team, 2024)

$BB(6) \geq \underbrace{10^{10^{\cdot^{\cdot^{\cdot^{10}}}}}}_{15}$

- Scott Aaronson: “The Busy Beaver Frontier”, [1].
- —“Why Philosophers Should Care About Computational Complexity?” [2]

Elementary Recursion Theory

An unsolvable problem

The diagonalisation language:

$$L_d := \{w \mid w = \langle M \rangle, w \notin L(M)\} = \{\langle M \rangle \mid \langle M \rangle \notin L(M)\}$$

An unsolvable problem

The diagonalisation language:

$$L_d := \{w \mid w = \langle M \rangle, w \notin L(M)\} = \{\langle M \rangle \mid \langle M \rangle \notin L(M)\}$$

Proposition

L_d is not recursively enumerable.

An unsolvable problem

The diagonalisation language:

$$L_d := \{w \mid w = \langle M \rangle, w \notin L(M)\} = \{\langle M \rangle \mid \langle M \rangle \notin L(M)\}$$

Proposition

L_d is not recursively enumerable.

Proof.

By diagonalisation. □

An unsolvable problem

The diagonalisation language:

$$L_d := \{w \mid w = \langle M \rangle, w \notin L(M)\} = \{\langle M \rangle \mid \langle M \rangle \notin L(M)\}$$

Proposition

L_d is not recursively enumerable.

Proof.

By diagonalisation. □

Membership in the diagonalisation language
Instance: w a binary word.
Question: Does $w \in L_d$ hold? (Does Tm. M with $\langle M \rangle = w$ accept w?)

An unsolvable problem

The diagonalisation language:

$$L_d := \{w \mid w = \langle M \rangle, w \notin L(M)\} = \{\langle M \rangle \mid \langle M \rangle \notin L(M)\}$$

Proposition

L_d is not recursively enumerable.

Proof.

By diagonalisation. □

Membership in the diagonalisation language
Instance: w a binary word.
Question: Does $w \in L_d$ hold? (Does Tm. M with $\langle M \rangle = w$ accept w?)

Theorem

There exist unsolvable decision problems.

Exercise: Halting Problem

Exercise

Try to adapt the diagonalisation argument to show that for the Halting Problem

$$H = \{ \langle \langle M \rangle, w \rangle \mid M \text{ halts on input } w \}$$

it holds:

- H is not recursive

and show that:

- H is recursively enumerable

Properties of r.e./recursive sets (I)

For $L \subseteq \Sigma^*$, $\bar{L} := \Sigma^* \setminus L$ is called the complement of L.

Proposition

If L is recursive, then $\bar{L}$ is recursive.

Properties of r.e./recursive sets (I)

For $L \subseteq \Sigma^*$, $\bar{L} := \Sigma^* \setminus L$ is called the complement of L.

Proposition

If L is recursive, then $\bar{L}$ is recursive.

Proof.

Let M be such that $L = L(M)$.

First idea: Swap the accepting states of M with the non-accepting states of M in which computations may halt.

Properties of r.e./recursive sets (I)

For $L \subseteq \Sigma^*$, $\bar{L} := \Sigma^* \setminus L$ is called the complement of L.

Proposition

If L is recursive, then $\bar{L}$ is recursive.

Proof.

Let M be such that $L = L(M)$.

First idea: Swap the accepting states of M with the non-accepting states of M in which computations may halt.

M is modified as follows to obtain $\bar{M}$:

- the accepting states of M are made non-accepting in $\bar{M}$.
- $\bar{M}$ has a new accepting state r.
- for each $q \in Q$ and tape symbol $s \in \Gamma$ such that $\delta_M(q, s)$ is undefined, add the transition $\delta_{\bar{M}}(q, s) = \langle r, s, R \rangle$.

Properties of r.e./recursive sets (I)

For $L \subseteq \Sigma^*$, $\bar{L} := \Sigma^* \setminus L$ is called the complement of L.

Proposition

If L is recursive, then $\bar{L}$ is recursive.

Proof.

Let M be such that $L = L(M)$.

First idea: Swap the accepting states of M with the non-accepting states of M in which computations may halt.

M is modified as follows to obtain $\bar{M}$:

- the accepting states of M are made non-accepting in $\bar{M}$.
- $\bar{M}$ has a new accepting state r.
- for each $q \in Q$ and tape symbol $s \in \Gamma$ such that $\delta_M(q, s)$ is undefined, add the transition $\delta_{\bar{M}}(q, s) = \langle r, s, R \rangle$.

It follows that $\bar{L} = L(\bar{M})$, and that $\bar{M}$ halts on all inputs. □

Properties of r.e./recursive sets (II)

Proposition

If both of L and $\bar{L}$ are r.e., then L is recursive.

Properties of r.e./recursive sets (II)

Proposition

If both of L and $\bar{L}$ are r.e., then L is recursive.

Proof.

Let M_1 and M_2 be Tm's such that $L = L(M_1)$ and $\bar{L} = L(M_2)$.

To decide, for a given $w \in \Sigma^*$, whether $w \in L$, build a Tm M that executes M_1 and M_2 on w in parallel, and such that:

- if M_1 accepts w, then also M accepts w.
- if M_2 accepts w, then also M halts, but does not accept w.

Hence M accepts w iff $w \in L(M_1) = L$. Thus $L(M) = L$.

Properties of r.e./recursive sets (II)

Proposition

If both of L and $\bar{L}$ are r.e., then L is recursive.

Proof.

Let M_1 and M_2 be Tm's such that $L = L(M_1)$ and $\bar{L} = L(M_2)$.

To decide, for a given $w \in \Sigma^*$, whether $w \in L$, build a Tm M that executes M_1 and M_2 on w in parallel, and such that:

- if M_1 accepts w, then also M accepts w.
- if M_2 accepts w, then also M halts, but does not accept w.

Hence M accepts w iff $w \in L(M_1) = L$. Thus $L(M) = L$.

Since for all w, either $w \in L$ or $w \in \bar{L}$, it follows that either M_1 or M_2 halts on w, and hence M halts on all inputs.

Hence $L = L(M)$ is recursive.

Exercise: Halting Problem

Exercise

Try to adapt the diagonalisation argument to show that for the Halting Problem

$$H = \{ \langle \langle M \rangle, w \rangle \mid M \text{ halts on input } w \}$$

it holds:

- H is not recursive

and show that:

- H is recursively enumerable

Universal language

The universal language:

$$L_u := \{\langle \langle M \rangle, w \rangle \mid w \in L(M)\}$$

Universal language

The universal language:

$$L_u := \{\langle \langle M \rangle, w \rangle \mid w \in L(M)\}$$

Theorem

L_u is recursively enumerable, but not recursive.

Universal language

The universal language:

$$L_u := \{\langle \langle M \rangle, w \rangle \mid w \in L(M)\}$$

Theorem

L_u is recursively enumerable, but not recursive.

Proof.

- L_u is r.e.: $L_u = L(M_u)$ for an universal machine M_u.

Universal language

The universal language:

$$L_u := \{\langle \langle M \rangle, w \rangle \mid w \in L(M)\}$$

Theorem

L_u is recursively enumerable, but not recursive.

Proof.

- L_u is r.e.: $L_u = L(M_u)$ for an universal machine M_u.
- L_u is not recursive:

  Suppose that L_u is recursive. Then $\bar{L_u}$ is recursive, and hence there exists a Tm. M such that $\bar{L_u} = L(M)$.

  M can be used to build a Tm. M' that accepts the diagonalisation language L_d, entailing $L_u = L(M')$.

  [picture of M' to be given]

  But then L_u would actually be r.e., in contradiction with what we proved before.

Busy Beaver is not computable

Definition (Tibor Radó, 1962)

$BB(n)$:= the largest number of steps any n-state Turing machine with tape alphabet $\{0, 1\}$ and blank symbol 0 can run before eventually halting, when started on an empty tape.

Proposition

The busy beaver function $BB : \mathbb{N} \to \mathbb{N},\ n \mapsto BB(n)$ *is not (Turing-)computable.*

More generally: one can solve the Halting Problem, given oracle access to any function $b : \mathbb{N} \to \mathbb{N}$ *such that* $b(n) \geq BB(n)$ *for all* $n \in \mathbb{N}$.

Busy Beaver is not computable

Definition (Tibor Radó, 1962)

$BB(n)$:= the largest number of steps any n-state Turing machine with tape alphabet $\{0, 1\}$ and blank symbol 0 can run before eventually halting, when started on an empty tape.

Proposition

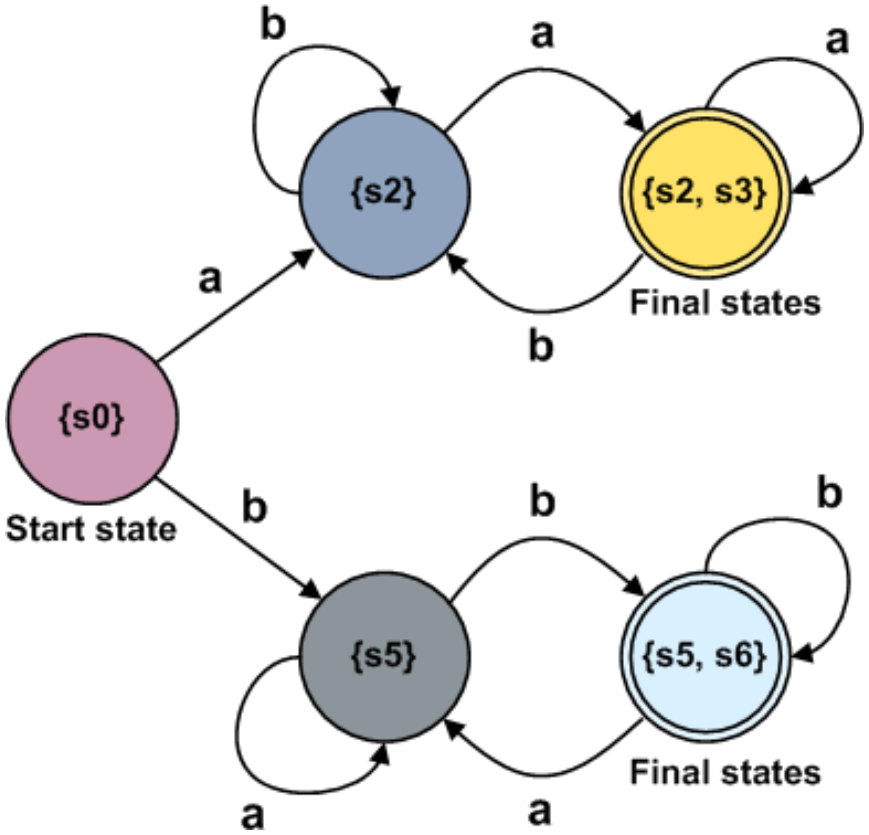
The busy beaver function $BB : \mathbb{N} \to \mathbb{N},\ n \mapsto BB(n)$ *is not (Turing-)computable.*

More generally: one can solve the Halting Problem, given oracle access to any function $b : \mathbb{N} \to \mathbb{N}$ *such that* $b(n) \geq BB(n)$ *for all* $n \in \mathbb{N}$.

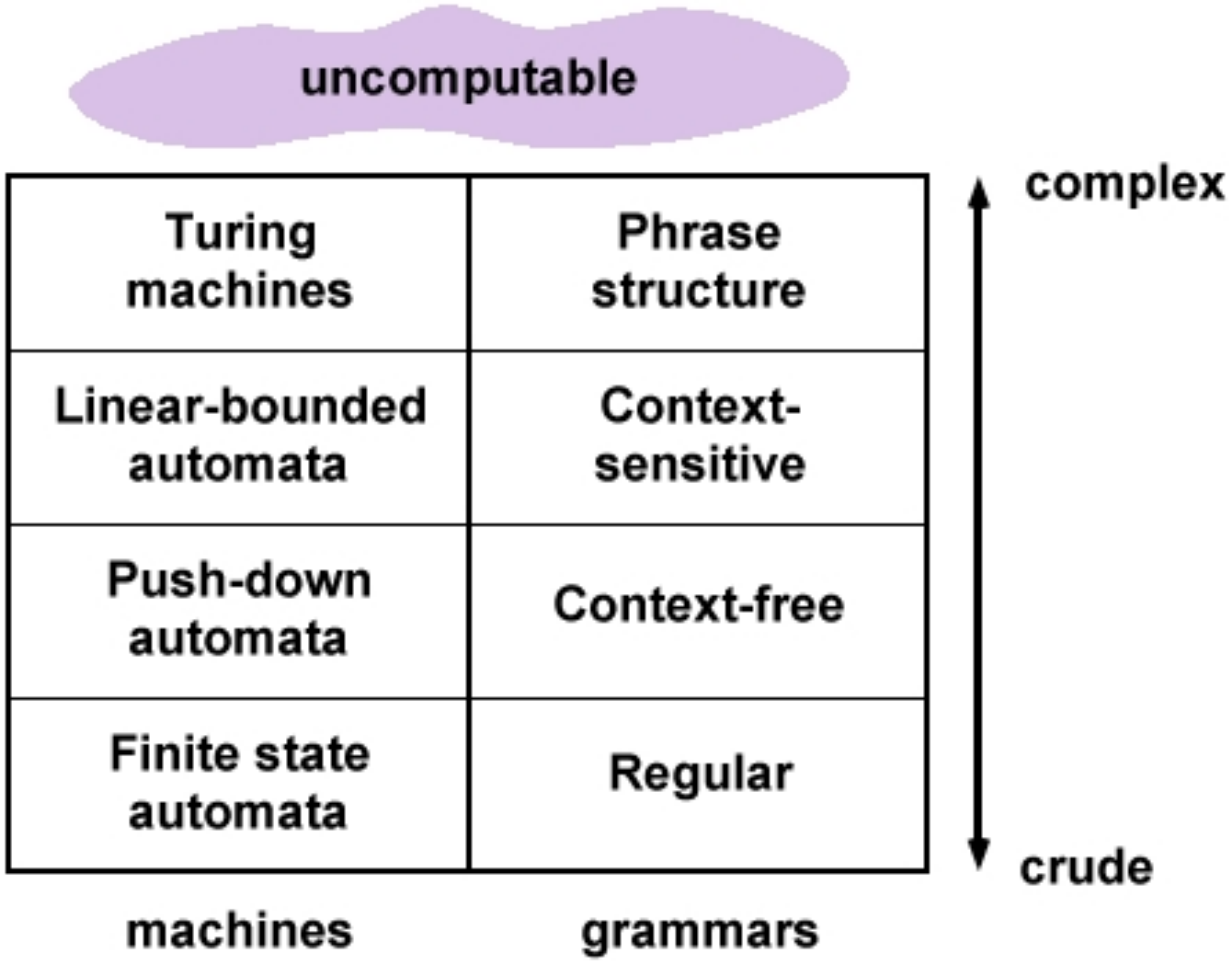
Proposition

Let $f : \mathbb{N} \to \mathbb{N}$ *be a Turing-computable function.*
Then there exists an $n_f \in \mathbb{N}$ *such that* $BB(n) > f(n)$ *for all* $n \geq n_f$.

Finite-state automaton



Formal-languages Chomsky hierarchy



Summary

- Post machine
- Turing machine
  - Turing's analysis of computations done by (human) computers
  - formal definition
  - video
  - typical features of models of computation
  - Busy Beavers
- Elementary recursion theory
  - an unsolvable problem
  - Halting problem
  - recursively enumerable, and recursive sets
  - universal language
  - Chomsky hierarchy

Course overview

Monday, July 6 10.30 – 12.30	Tuesday, July 7 10.30 – 12.30	Wednesday, July 8 10.30 – 12.30	Thursday, July 9 10.30 – 12.30	Friday, July 10
intro	*classic models*			*additional models*
Introduction to Computability	**Machine Models**	**Recursive Functions**	**Lambda Calculus**	
computation and decision problems, from logic to computability, overview of models of computation relevance of MoCs	Post Machines, typical features, Turing's analysis of human computers, Turing machines, basic recursion theory	primitive recursive functions, Gödel–Herbrand recursive functions, partial recursive funct's, partial recursive = = Turing-computable, Church's Thesis	λ-terms, β-reduction, λ-definable functions, partial recursive = λ-definable = Turing computable	
	imperative programming	*algebraic programming*	*functional programming*	
				14.30 – 16.30
				Three more Models of Computation
				Post's Correspondence Problem, Interaction-Nets, Fractran comparing computational power

References I

Scott Aaronson.
The Busy Beaver Frontier.
SIGACT News, 51(3):32–54, Sept 2020.

Scott Aaronson.
Why Philosophers Should Care About Computational Complexity.
talk at the University of Texas at Austin, May 28, 2025.
on youtube at `https://youtu.be/OST1DjD08Hg?si=PS9O0x3szKF7LhSZ`.

Maribel Fernández.
Models of Computation (An Introduction to Computability Theory).
Springer, Dordrecht Heidelberg London New York, 2009.

References II

Emil Leon Post.
Finite Combinatory Processes – Formulation 1.
Journal of Symbolic Logic, 1(3):103–105, 1936.
https://www.wolframscience.com/prizes/tm23/images/Post.pdf.

Alan M. Turing.
On Computable Numbers, with an Application to the Entscheidungsproblem.
Proceedings of the London Mathematical Society, 42(2):230–265, 1936.
http://www.wolframscience.com/prizes/tm23/images/Turing.pdf.