

# Lecture 2: Machine Models, Basic Computability Theory Models of Computation

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

Clemens Grabmayer

Teaching Mobility Program (PNRR-TNE DESK)  
University of Novi Sad  
Novi Sad, Serbia

March ., 2026

# Course overview

<i>intro</i>	<i>classic models</i>			<i>additional models</i>
<b>Introduction to Computability</b>	<b>Machine Models</b>	<b>Recursive Functions</b>	<b>Lambda Calculus</b>	<b>Three more Models of Computation</b>
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	$\lambda$ -terms, $\beta$ -reduction, $\lambda$ -definable functions, partial recursive = $\lambda$ -definable = Turing computable	Post's Correspondence Problem, Interaction-Nets, Fractran
	<i>imperative programming</i>	<i>algebraic programming</i>	<i>functional programming</i>	

# Overview

- ▶ exercise calculable predicate

# Overview

- ▶ exercise calculable predicate
- ▶ Post machine
- ▶ Turing machine
  - ▶ Turing's analysis of computations done by (human) computers
  - ▶ formal definition
  - ▶ video

# Overview

- ▶ exercise calculable predicate
- ▶ Post machine
- ▶ Turing machine
  - ▶ Turing's analysis of computations done by (human) computers
  - ▶ formal definition
  - ▶ video
- ▶ Elementary recursion theory
  - ▶ an unsolvable problem
  - ▶ Halting problem
  - ▶ recursively enumerable, and recursive sets
  - ▶ universal language
  - ▶ Chomsky hierarchy

# Calculable functions?

## Questions/Exercises

- ① Suppose  $P(a, b)$  is a calculable predicate.  
Why does  $(\exists x)P(a, x)$  not have to be calculable?
- ② Let  $f : \mathbb{N} \rightarrow \mathbb{N}$  defined by

$$n \mapsto \begin{cases} 0 & \dots n = 0 \text{ \& Goldbach's conjecture is false} \\ 1 & \dots n = 0 \text{ \& Goldbach's conjecture is true} \\ n+1 & \dots n > 0 \end{cases}$$

Is  $f$  calculable?

- ③ Can computation problems for mappings  $F : \mathbb{N}^n \rightarrow \mathbb{N}^m$  always be represented by decision problems?

# Calculable functions?

## Questions/Exercises

- ① Suppose  $P(a, b)$  is a calculable predicate.  
Why does  $(\exists x)P(a, x)$  not have to be calculable?

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

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**  $a \in A$ .

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

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**  $a \in A$ .

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  $E$ .

# Reading recommended (for today)

① Post machine: Page 1 + first paragraph on page 2 of:

- ▶ Emil Post: *Finite Combinatory Processes – Formulation 1*, Journal of Symbolic Logic (1936), [2].

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), [3].

# Emil Post



Emil Leon Post (1897–1954)

# Post about ...

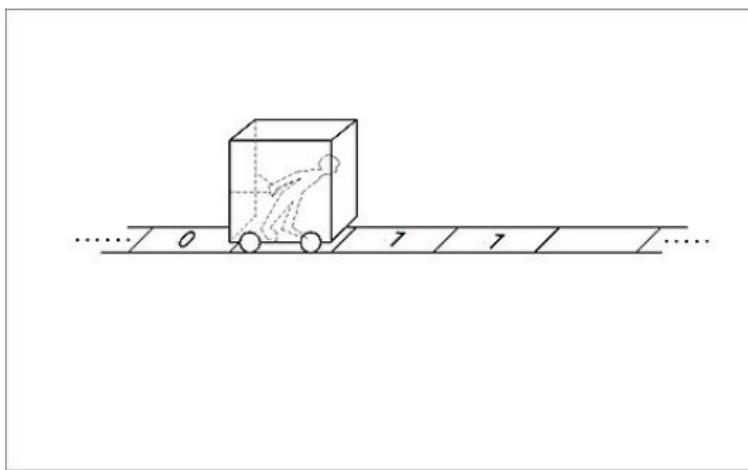
... a result of his from 1921 similar to the Incompleteness Theorem:

Theorem (Gödel, 1931 (paraphrased here))

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

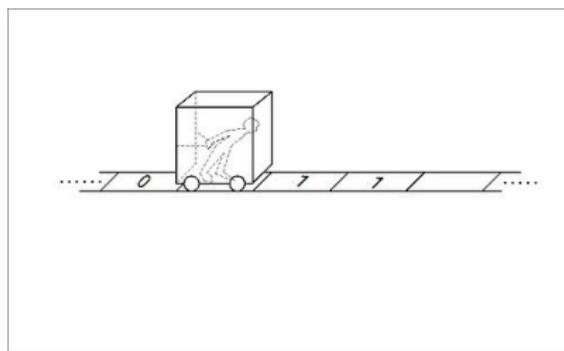
*"For full generality a complete analysis would have to be given of all possible ways in which the human mind could set up finite processes for generating sequences."*

# Post machine (1936)



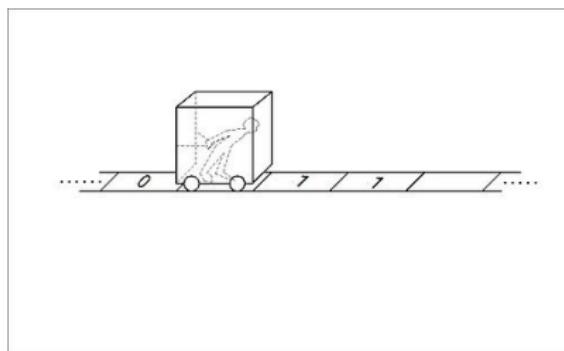
Emil Post: *Finite Combinatory Processes – Formulation 1* (1936),  
Journal of Symbolic Logic, [2].

# Post machine (1936)



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

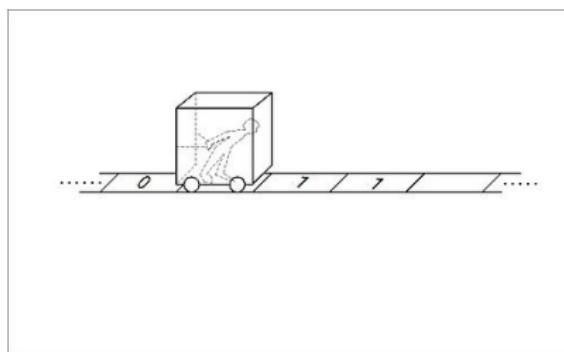
# Post machine (1936)



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

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

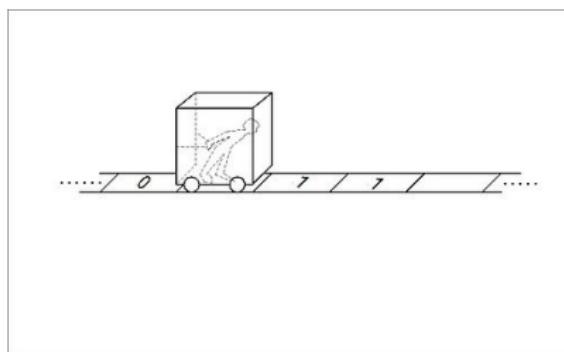
# Post machine (1936)



“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),

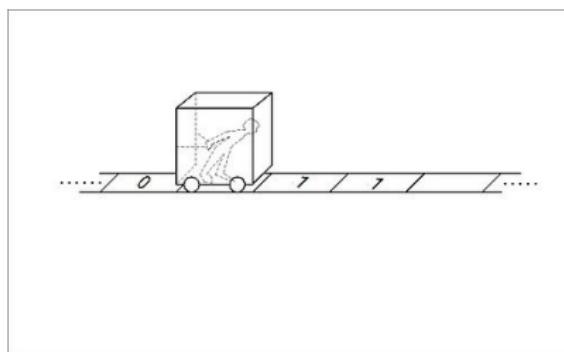
# Post machine (1936)



“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,

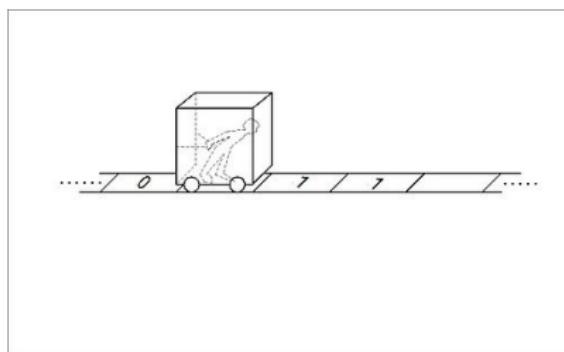
# Post machine (1936)



“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,

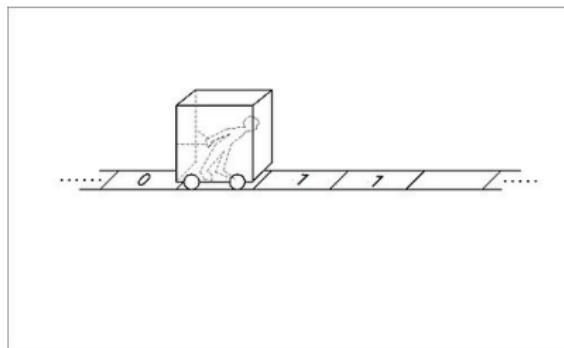
# Post machine (1936)



“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.”

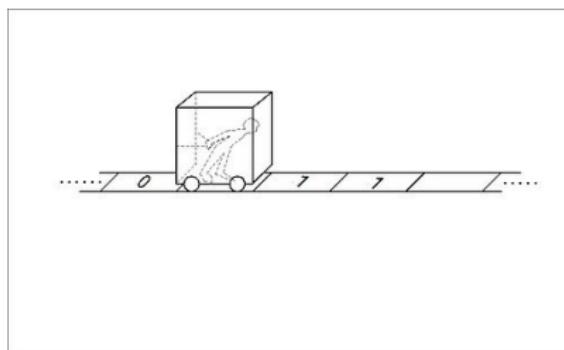
# Post machine (1936)



'Directions' (= list of instructions):

- ▶ Start at the starting point and follow direction 1.

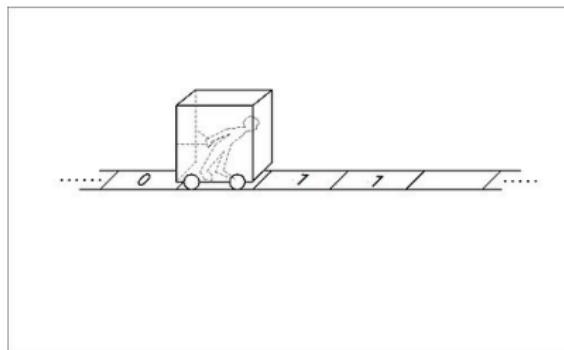
# Post machine (1936)



'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:

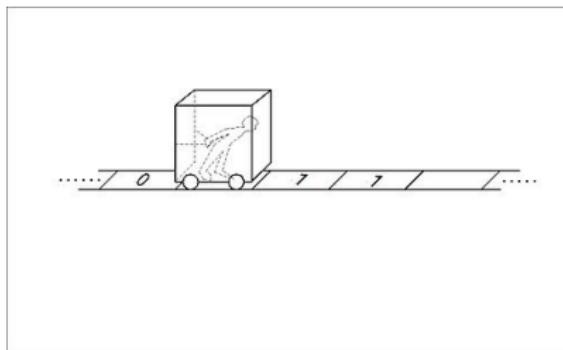
# Post machine (1936)



'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  $j_i$ .

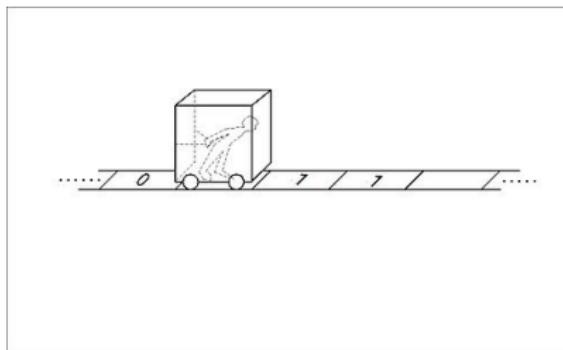
# Post machine (1936)



'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  $j''_i$ .

# 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)

(Credits due to: [Vincent van Oostrom](#))

# Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)

(Credits due to: [Vincent van Oostrom](#))

# Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)
- ▶ modification

(Credits due to: [Vincent van Oostrom](#))

# Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)
- ▶ modification
  - ▶ of (immediately accessible) stored data

(Credits due to: [Vincent van Oostrom](#))

# Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)
- ▶ modification
  - ▶ of (immediately accessible) stored data
  - ▶ of control state

(Credits due to: [Vincent van Oostrom](#))

# Typical features of ‘computationally complete’ MoC’s

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

(Credits due to: [Vincent van Oostrom](#))

# Typical features of ‘computationally complete’ MoC’s

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

(Credits due to: [Vincent van Oostrom](#))

# 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's analysis of a human 'computer'

Section 9 in Turing's 1937 paper 'On computable numbers, with an application to the Entscheidungsproblem' [3].

A direct appeal to intuition in analysing human computation:

- ▶ paper is divided into squares

# Turing's analysis of a human 'computer'

Section 9 in Turing's 1937 paper 'On computable numbers, with an application to the Entscheidungsproblem' [3].

A direct appeal to intuition in analysing human computation:

- ▶ paper is divided into squares
- ▶ one-dimensional paper ('tape' divided into squares)

# Turing's analysis of a human 'computer'

Section 9 in Turing's 1937 paper 'On computable numbers, with an application to the Entscheidungsproblem' [3].

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

# Turing's analysis of a human 'computer'

Section 9 in Turing's 1937 paper 'On computable numbers, with an application to the Entscheidungsproblem' [3].

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:

# Turing's analysis of a human 'computer'

Section 9 in Turing's 1937 paper 'On computable numbers, with an application to the Entscheidungsproblem' [3].

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

# Turing's analysis of a human 'computer'

Section 9 in Turing's 1937 paper 'On computable numbers, with an application to the Entscheidungsproblem' [3].

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'

# Turing's analysis of a human 'computer'

Section 9 in Turing's 1937 paper 'On computable numbers, with an application to the Entscheidungsproblem' [3].

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

# Turing's analysis of a human 'computer'

Section 9 in Turing's 1937 paper 'On computable numbers, with an application to the Entscheidungsproblem' [3].

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."*

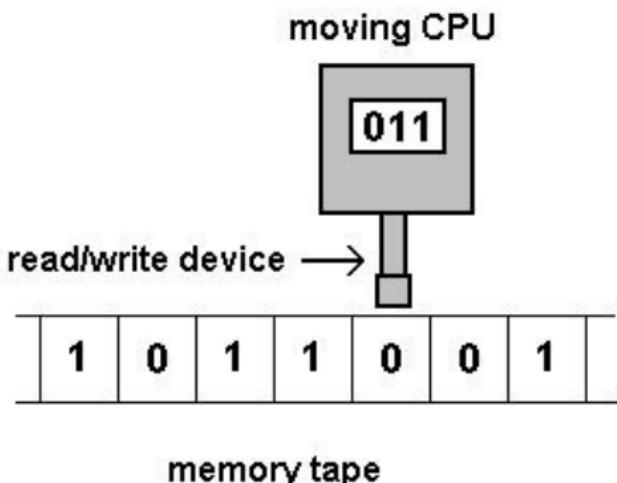
# Turing's analysis of a human 'computer'

- ▶ 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



memory tape

# 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  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \emptyset, F \rangle$  where:

## Turing machine: formal definition

## Definition

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

- ▶  $Q$  is a finite set of **states**;
- ▶  $\Sigma$  is the **input alphabet**;
- ▶  $\Gamma$  is the **tape alphabet** that is finite and  $\Gamma \supseteq \Sigma \cup \{\#\}$  holds;
  
- ▶  $\#$  is a designated **blank symbol** not contained in  $\Sigma$ ;
- ▶  $q_0 \in Q$  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  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \# \rangle$  where:

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

# Turing machine: definition notions

## Definition

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

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

# Turing machine: definition notions

## Definition

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

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

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

# Turing machine: definition notions

## Definition

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

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

- ▶  $uqav'$  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  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \#\#, F \rangle$  be a Turing machine.

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

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

$\vdash_M \dots$  next-move-relation

$\vdash_M^* \dots$  reflexive, and transitive closure of  $\vdash_M$

# Turing machine: definition notions

## Definition

Let  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \mathbb{b}, F \rangle$  be a Turing machine.

A **configuration** of  $M$  is elements  $w_1 q w_2 \in \Gamma^* \times Q \times \Gamma^*$  such that the first letter in  $w_1$  and the last letter in  $w_2$  are different from  $\mathbb{b}$ .

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

$\vdash_M \dots$  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  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \mathbb{b}, F \rangle$  be a Turing machine.

A **configuration** of  $M$  is elements  $w_1 q w_2 \in \Gamma^* \times Q \times \Gamma^*$  such that the first letter in  $w_1$  and the last letter in  $w_2$  are different from  $\mathbb{b}$ .

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

$\vdash_M \dots$  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  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \mathbb{b}, F \rangle$  be a Turing machine.

A **configuration** of  $M$  is elements  $w_1 q w_2 \in \Gamma^* \times Q \times \Gamma^*$  such that the first letter in  $w_1$  and the last letter in  $w_2$  are different from  $\mathbb{b}$ .

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

$\vdash_M \dots$  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/recursive languages

## Definition

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

$L$  is called **recursively enumerable** if

- ▶  $L = L(M)$  for some Turing machine  $M$  with input symbols  $\Sigma$ .

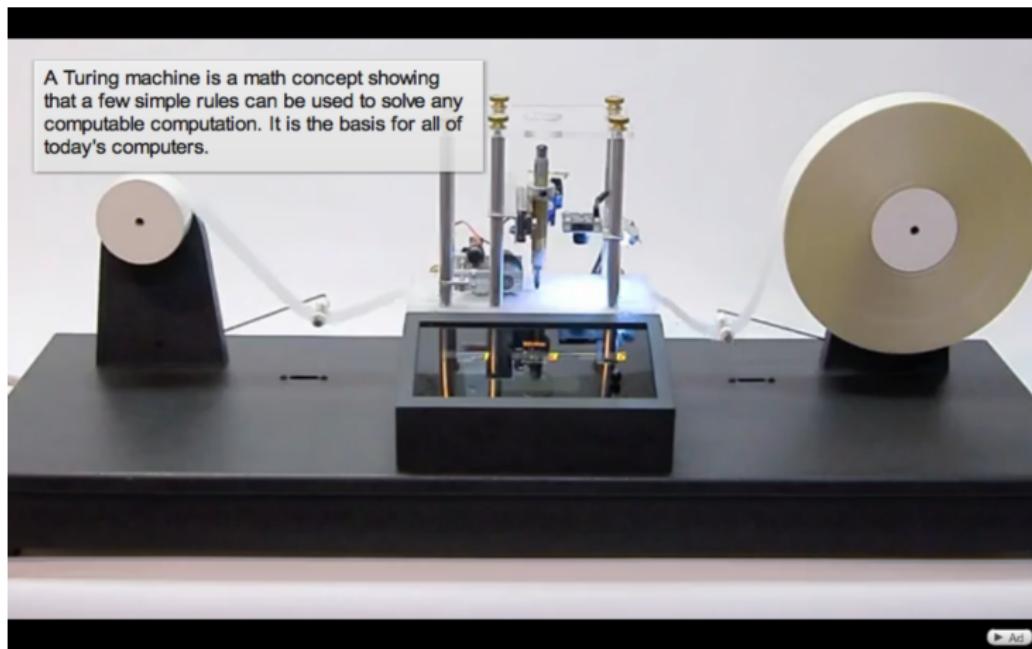
$L$  is called **recursive** if

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

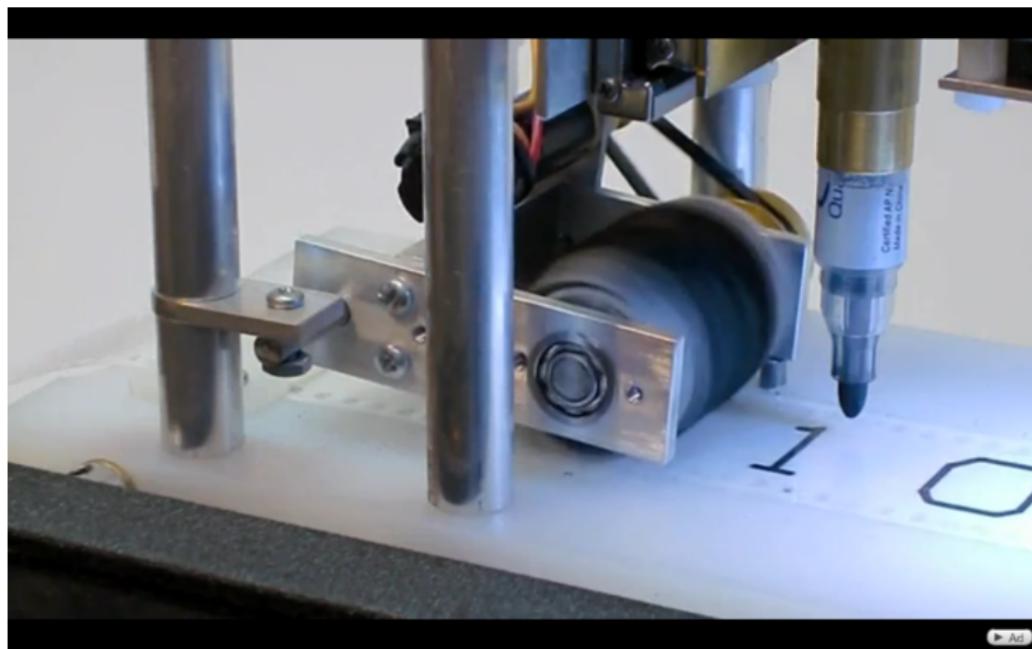
# Mike Davey's Turing machine ([link](#))



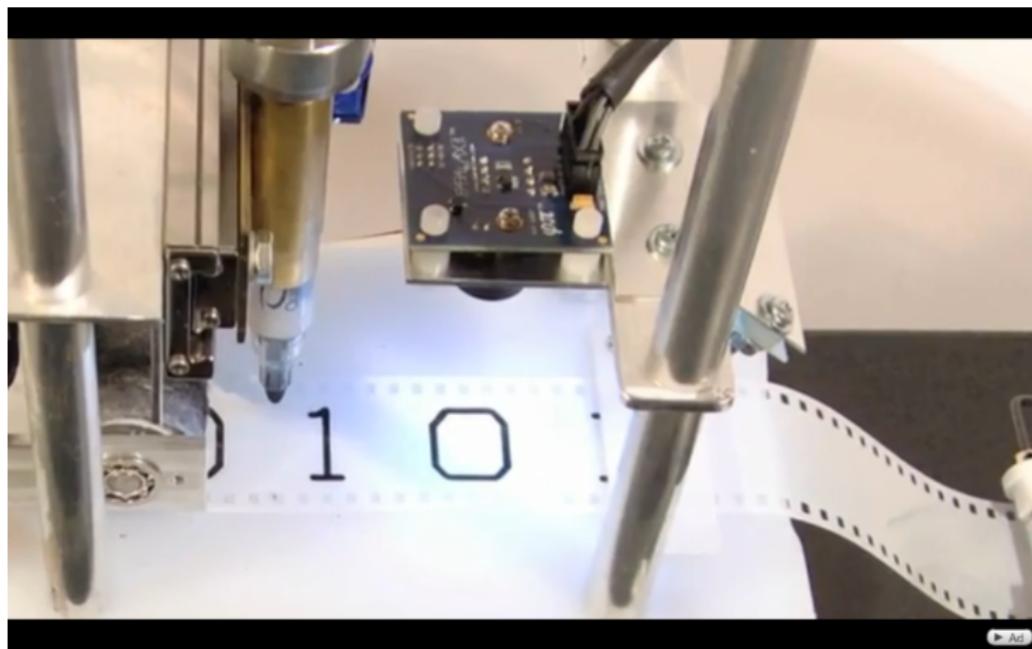
# Mike Davey's Turing machine ([link](#))



# Mike Davey's Turing machine ([link](#))



# Mike Davey's Turing machine ([link](#))



# Mike Davey's Turing machine ([link](#))



# 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

010110111011110111110...

on the tape, but does not halt.

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

# Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)

# Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)

# Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)
- ▶ modification

# Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)
- ▶ modification
  - ▶ of (immediately accessible) stored data

# Typical features of ‘computationally complete’ MoC’s

- ▶ storage (unbounded)
- ▶ control (finite, given)
- ▶ modification
  - ▶ of (immediately accessible) stored data
  - ▶ of control state

# Typical features of ‘computationally complete’ MoC’s

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

# Typical features of ‘computationally complete’ MoC’s

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

# 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)
- ▶ TM's with multiple tapes
  - ▶ Input/Output Turing machines (with input- and output tapes)
- ▶ non-deterministic TM's:  $\delta \subseteq ((Q \times \Gamma) \times (Q \times \Gamma \times \{\text{L}, \text{R}\}))$
- ▶ 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

# 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:

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

# Universal language

The universal language:

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

Theorem

$\textcolor{brown}{L}_u$  is recursively enumerable, but not recursive.

# Universal language

The universal language:

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

Theorem

$\textcolor{brown}{L}_u$  is recursively enumerable, but not recursive.

Proof.

- ▶  $\textcolor{brown}{L}_u$  is r.e.:  $\textcolor{brown}{L}_u = L(M_u)$  for an universal machine  $M_u$ .

# Universal language

The universal language:

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

Theorem

$\textcolor{brown}{L}_u$  is recursively enumerable, but not recursive.

Proof.

- ▶  $\textcolor{brown}{L}_u$  is r.e.:  $\textcolor{brown}{L}_u = L(M_u)$  for an universal machine  $M_u$ .
- ▶  $\textcolor{brown}{L}_u$  is not recursive:

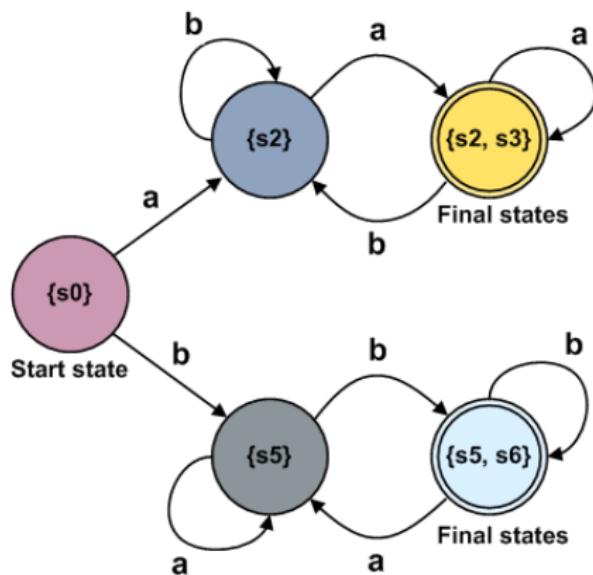
Suppose that  $\textcolor{brown}{L}_u$  is recursive. Then  $\bar{L}_u$  is recursive, and hence there exists a Tm.  $\textcolor{violet}{M}$  such that  $\bar{L}_u = L(\textcolor{violet}{M})$ .

$\textcolor{violet}{M}$  can be used to build a Tm.  $\textcolor{red}{M}'$  that accepts the diagonalisation language  $\textcolor{green}{L}_d$ , entailing  $\textcolor{brown}{L}_u = L(\textcolor{red}{M}')$ .

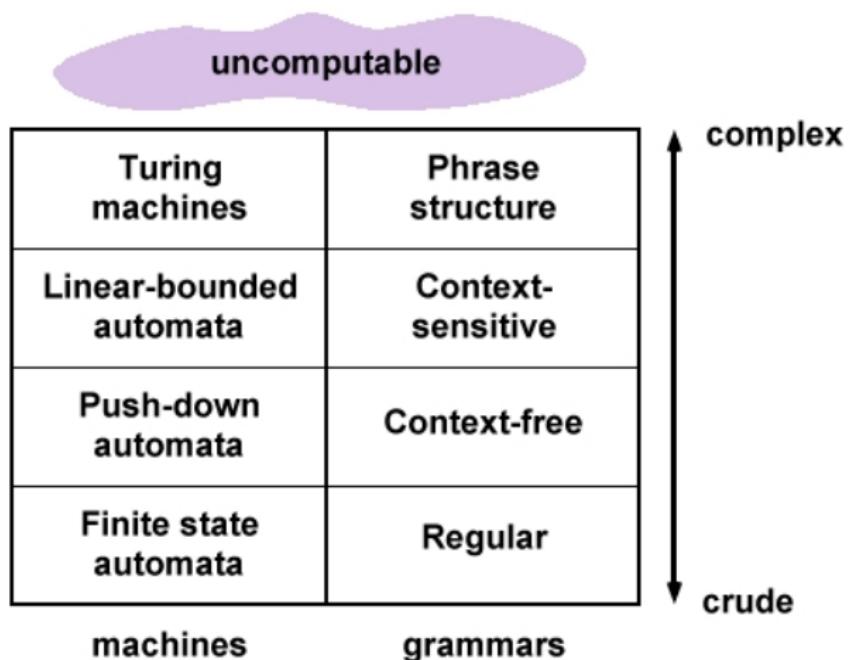
[picture of  $\textcolor{red}{M}'$  to be given]

But then  $\textcolor{brown}{L}_u$  would actually be r.e., in contradiction with what we proved before.

# Finite-state automaton



# Formal-languages Chomsky hierarchy



# Summary

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

# Recommended reading

## ① Recursive and primitive-recursive functions:

Chapter 4, Recursive Functions of the book:

- ▶ Maribel Fernández [1]: *Models of Computation (An Introduction to Computability Theory)*, Springer-Verlag London, 2009.

# Course overview

<i>intro</i>	<i>classic models</i>			<i>additional models</i>
<b>Introduction to Computability</b>	<b>Machine Models</b>	<b>Recursive Functions</b>	<b>Lambda Calculus</b>	<b>Three more Models of Computation</b>
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	$\lambda$ -terms, $\beta$ -reduction, $\lambda$ -definable functions, partial recursive = $\lambda$ -definable = Turing computable	Post's Correspondence Problem, Interaction-Nets, Fractran
	<i>imperative programming</i>	<i>algebraic programming</i>	<i>functional programming</i>	

# References

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