Lecture 2: Machine Models, Basic Computability Theory

Models of Computation

Clemens Grabmayer

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

July 8, 2025

Course overview

Monday, July 7 10.30 – 12.30	Tuesday, July 8 10.30 – 12.30	Wednesday, July 9 10.30 – 12.30	Thursday, July 10 10.30 – 12.30	Friday, July 11
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

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 . . .

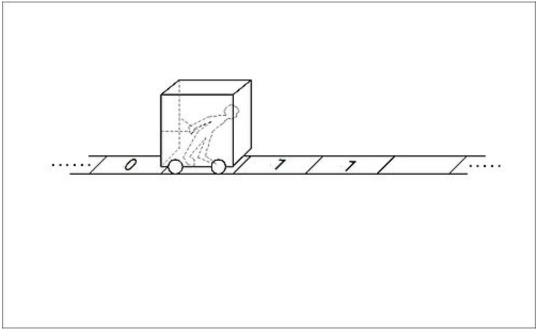
. . . a result of his from 1921 similar to the Incompleteness Theorem:

Theorem (Gödel, 1931 (paraphrased here))

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

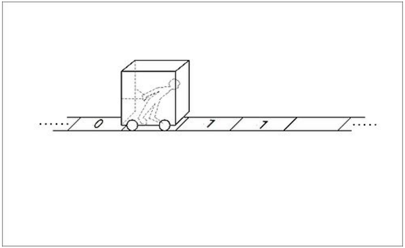
> *"For full generality a complete analysis would have to be given of all possible ways in which the human mind could set up finite processes for generating sequences."*

Post machine (1936)



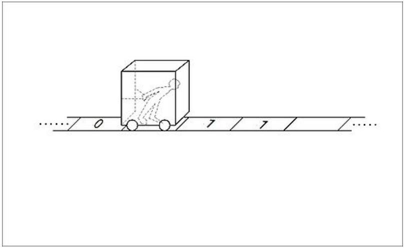
Emil Post: *Finite Combinatory Processes – Formulation 1* (1936), Journal of Symbolic Logic, [2].

Post machine (1936)



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

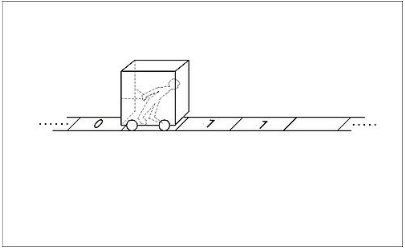
Post machine (1936)



"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)

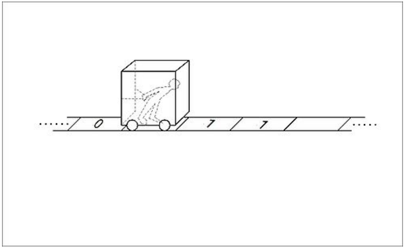


“The worker is assumed to be capable of performing the following primitive acts:

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

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

Post machine (1936)



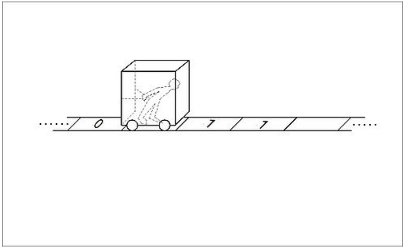
"The worker is assumed to be capable of performing the following primitive acts:

(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,

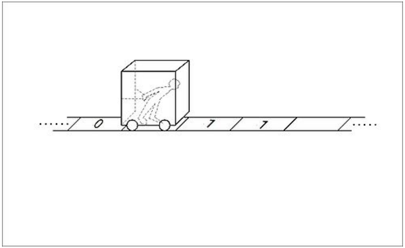
Post machine (1936)



"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,

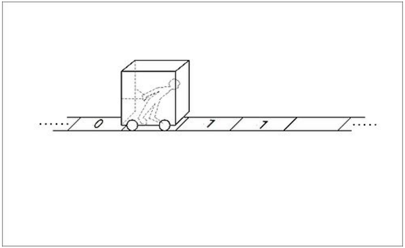
Post machine (1936)



“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.”

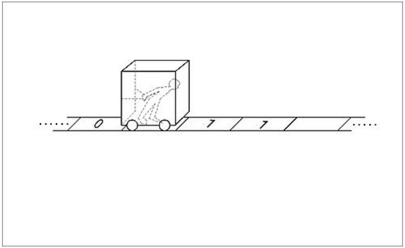
Post machine (1936)



‘Directions’ (instructions):

- Start at the starting point and follow direction 1.

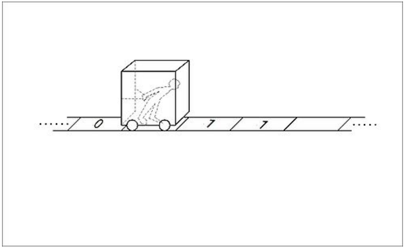
Post machine (1936)



'Directions' (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:

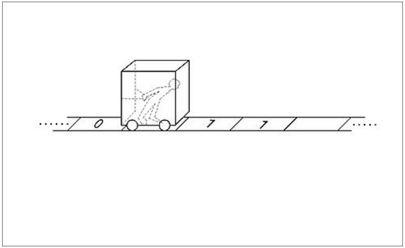
Post machine (1936)



‘Directions’ (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:
  (A) Perform operation $O_i \in \{(a), (b), (c), (d)\}$, then follow direction j_i.

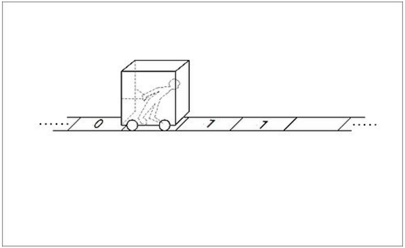
Post machine (1936)



'Directions' (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:
  - (A) Perform operation $O_i \in \{(\text{a}), (\text{b}), (\text{c}), (\text{d})\}$, then follow direction j_i.
  - (B) Perform operation (e) and according as the answer is yes or no correspondingly follow direction j_i' or j_i''.

Post machine (1936)



‘Directions’ (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:
  - (A) Perform operation $O_i \in \{$(a), (b), (c), (d)$\}$, then follow direction j_i.
  - (B) Perform operation (e) and according as the answer is yes or no correspondingly follow direction j_i' or j_i''.
  - (C) 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 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 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
  - 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 symbol on one of the observed squares
  - (b) change of one of the squares observed to another square within L squares of a previously observed one.

Turing's analysis of a human 'computer'

- simple operations must include:
  (a) change of a symbol on one of the observed squares
  (b) change of one of the squares observed to another square within L squares of a previously observed one.
- most general simple operations:

Turing's analysis of a human 'computer'

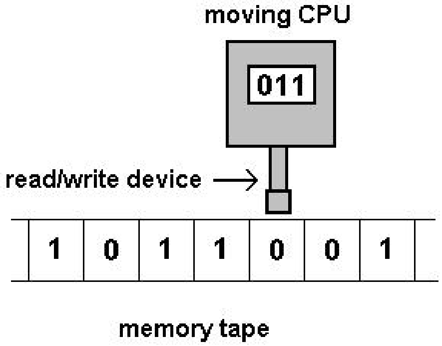
- simple operations must include:
  (a) change of a symbol on one of the observed squares
  (b) change of one of the squares observed to another square within L squares of a previously observed one.
- most general simple operations:
  (A) A change (a) of symbol with a possible change of state of mind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

"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 $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not{b}, F \rangle$ where:

- Q is a finite set of states;

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;
- Σ is the input alphabet;

Turing machine: formal definition

Definition

A Turing machine is a tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not{b}, F \rangle$ where:

- Q is a finite set of states;
- Σ is the input alphabet;
- Γ is the tape alphabet that is finite and $\Gamma \supseteq \Sigma \cup \{\not{b}\}$ holds;

Turing machine: formal definition

Definition

A Turing machine is a tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not{b}, F \rangle$ where:

- Q is a finite set of states;
- Σ is the input alphabet;
- Γ is the tape alphabet that is finite and $\Gamma \supseteq \Sigma \cup \{\not{b}\}$ holds;
- $\delta : (Q \smallsetminus F) \times \Gamma \rightharpoonup Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is a partial function, called the transition function;

Turing machine: formal definition

Definition

A Turing machine is a tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not{b}, F \rangle$ where:

- Q is a finite set of states;
- Σ is the input alphabet;
- Γ is the tape alphabet that is finite and $\Gamma \supseteq \Sigma \cup \{\not{b}\}$ holds;
- $\delta : (Q \smallsetminus F) \times \Gamma \rightharpoonup Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is a partial function, called the transition function;
- $\not{b}$ is a designated blank symbol not contained in Σ;

Turing machine: formal definition

Definition

A Turing machine is a tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not{b}, F \rangle$ where:

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

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;
- Σ is the input alphabet;
- Γ is the tape alphabet that is finite and $\Gamma \supseteq \Sigma \cup \{b\!\!\!/\}$ holds;
- $\delta : (Q \smallsetminus F) \times \Gamma \rightharpoonup Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is a partial function, called the transition function;
- $b\!\!\!/$ 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 $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not b, F \rangle$ 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 $\not b$.

Turing machine: definition notions

Definition

Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, b\!\!\!/, F \rangle$ 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 $b\!\!\!/$.

- $uqav'$ 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 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 $\not{b}$.

- $uqav'$ 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 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 $\not{b}$.

- $uqav'$ 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 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 $\not{b}$.

- $uqav'$ 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 $M = \langle Q, \Sigma, \Gamma, \delta, q_0, b\!\!\!/, F \rangle$ 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 $b\!\!\!/$.

- $uqav'$ 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 $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \not b, F \rangle$ 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 $\not b$.

- $uqav'$ 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.

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

Recursively enumerable/recursive languages

Definition

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

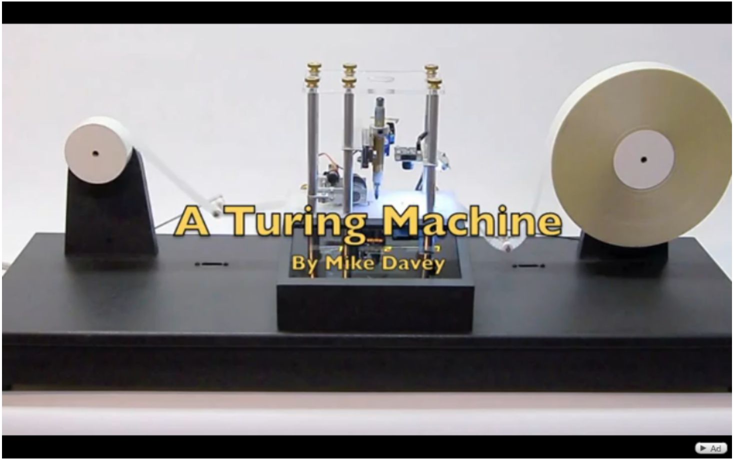
L is called recursively enumerable if

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

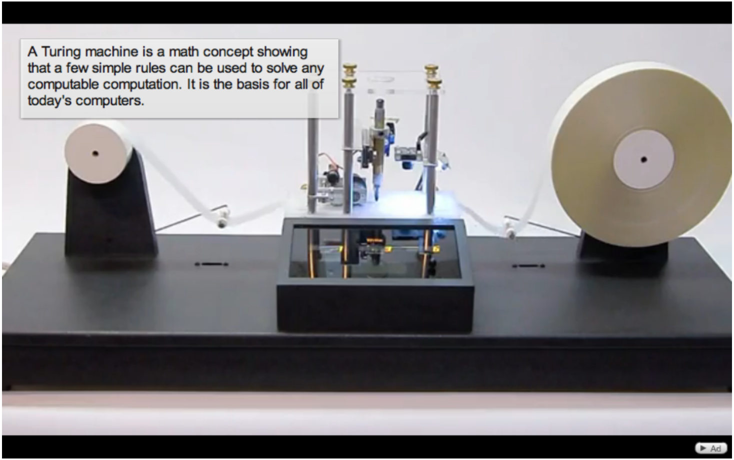
L is called recursive if

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

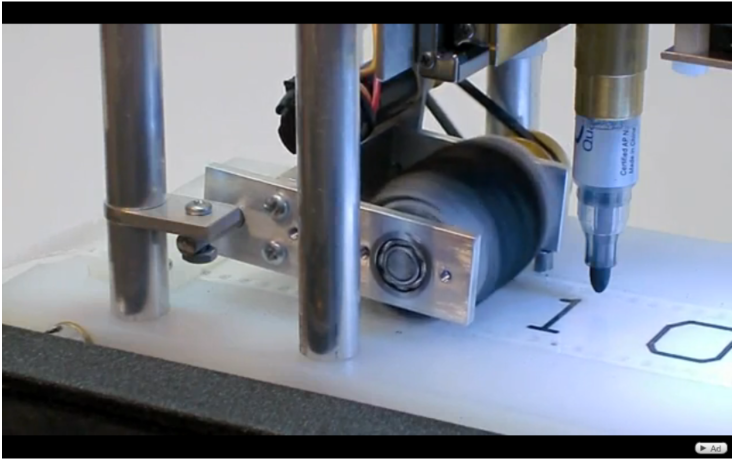
Mike Davey's Turing machine (link)



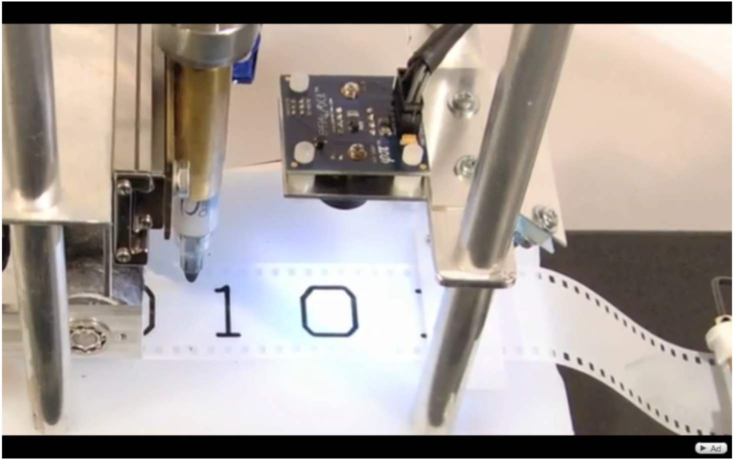
Mike Davey's Turing machine (link)



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

$$010110111011110111110\ldots$$

on the tape, but does not halt.

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

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

An unsolvable problem

The diagonalisation language:

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

An unsolvable problem

The diagonalisation language:

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

Proposition

L_d *is not recursively enumerable.*

An unsolvable problem

The diagonalisation language:

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

Proposition

L_d is not recursively enumerable.

Proof.

By diagonalisation. □

An unsolvable problem

The diagonalisation language:

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

Proposition

L_d is not recursively enumerable.

Proof.

By diagonalisation. $\square$

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:

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

Proposition

L_d is not recursively enumerable.

Proof.

By diagonalisation. $\square$

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

$$H = \{w \mid w = \langle w_n, w_m \rangle,\ M_n \text{ halts on input } w_m\}$$

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 $\bar{M}$:

1. the accepting states of M are made non-accepting in $\bar{M}$.
2. $\bar{M}$ has a new accepting state r.
3. 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 $\bar{M}$:

1. the accepting states of M are made non-accepting in $\bar{M}$.
2. $\bar{M}$ has a new accepting state r.
3. 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. $\square$

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

Proposition

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

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

Proposition

If both of L and $\bar{L}$ is 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 $\bar{L}$ is 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.

Universal language

The universal language:

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

Universal language

The universal language:

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

Theorem

L_u is r.e., but not recursive.

Universal language

The universal language:

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

Theorem

L_u is r.e., but not recursive.

Proof.

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

Universal language

The universal language:

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

Theorem

L_u is r.e., but not recursive.

Proof.

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

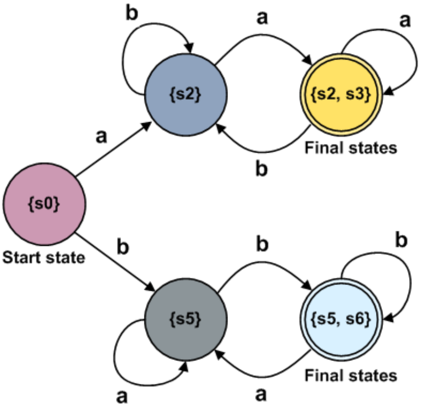
   Suppose that L_u is recursive. Then $\overline{L_u}$ is recursive, and hence there exists a Tm. M such that $\overline{L_u} = L(M)$.

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

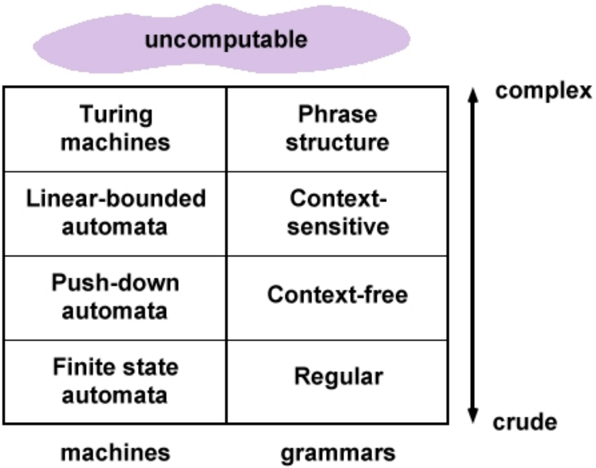
   [picture of M' to be given]

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

Finite-state automaton



Formal-languages Chomsky hierarchy



Summary

Example suggestions

Examples

1.
2.
3.

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

Monday, July 7 10.30 – 12.30	Tuesday, July 8 10.30 – 12.30	Wednesday, July 9 10.30 – 12.30	Thursday, July 10 10.30 – 12.30	Friday, July 11
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

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

Emil L. 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.