The notion of Turing machine

Peter Hajnal

Bolyai Institute, SZTE, Szeged

2023 fall

Characters, words

Characters, words

A computational problems is a function $f : \mathcal{I} \to \mathcal{O}$, where $\mathcal{I}$ is the set of inputs and $\mathcal{O}$ is the set of possible outputs.

Characters, words

A computational problems is a function $f : \mathcal{I} \to \mathcal{O}$, where $\mathcal{I}$ is the set of inputs and $\mathcal{O}$ is the set of possible outputs. We always assume that the input, and output are coded in a well-defined way.

Characters, words

A computational problems is a function $f : \mathcal{I} \to \mathcal{O}$, where $\mathcal{I}$ is the set of inputs and $\mathcal{O}$ is the set of possible outputs. We always assume that the input, and output are coded in a well-defined way.

Definition: Alphabet

Let Σ be a non-empty, finite set. The elements of Σ are called characters, Σ is refered as *alphabet*,

Characters, words

A computational problems is a function $f : \mathcal{I} \to \mathcal{O}$, where $\mathcal{I}$ is the set of inputs and $\mathcal{O}$ is the set of possible outputs. We always assume that the input, and output are coded in a well-defined way.

Definition: Alphabet

Let Σ be a non-empty, finite set. The elements of Σ are called characters, Σ is refered as *alphabet*,

Definition: Words

Let Σ e an alphabet. The elements ofΣ^ℓ are character sequenses of length ℓ. The elements of Σ^ℓ are called words of length ℓ. Σ^0 contains a single word, the empty word (ε), the unique word of length 0. Σ^* is the set containing the words of finite length over the alphabet Σ, i.e. $\Sigma^* = \cup_{i=0}^{\infty} \Sigma^i$.

Coding

Coding

Definition: Coding

The codings of the sets of inputs/outputs are two (1-1/injective) functions:

$$c_I : \mathcal{I} \to \Sigma^*, \qquad c_O : \mathcal{O} \to \Sigma^*.$$

Coding

Definition: Coding

The codings of the sets of inputs/outputs are two (1-1/injective) functions:

$$c_I : \mathcal{I} \to \Sigma^*, \qquad c_O : \mathcal{O} \to \Sigma^*.$$

To encode, you must first choose an alphabet. Then we could stick to the choice $\Sigma = \{0, 1\}$. However, sometimes it is technically easier to work with a larger alphabet.

Coding

Definition: Coding

The codings of the sets of inputs/outputs are two (1-1/injective) functions:

$$c_I : \mathcal{I} \to \Sigma^*, \qquad c_O : \mathcal{O} \to \Sigma^*.$$

To encode, you must first choose an alphabet. Then we could stick to the choice $\Sigma = \{0, 1\}$. However, sometimes it is technically easier to work with a larger alphabet.

Note: Note that only countable $\mathcal{I}$ and $\mathcal{O}$ can be encoded. That is, for set-theoretic reasons, the real numbers sets of real numbers are not encodable.

Coding

Definition: Coding

The codings of the sets of inputs/outputs are two (1-1/injective) functions:

$$c_I : \mathcal{I} \to \Sigma^*, \qquad c_O : \mathcal{O} \to \Sigma^*.$$

To encode, you must first choose an alphabet. Then we could stick to the choice $\Sigma = \{0, 1\}$. However, sometimes it is technically easier to work with a larger alphabet.

Note: Note that only countable $\mathcal{I}$ and $\mathcal{O}$ can be encoded. That is, for set-theoretic reasons, the real numbers sets of real numbers are not encodable.

Definition: Computational problem

$$f : \Sigma^* \to \Sigma^*.$$

The size of the input

The size of the input

After an encoding, we can talk about *the size of the input*: the number of characters in the input character sequence.

The size of the input

After an encoding, we can talk about *the size of the input*: the number of characters in the input character sequence.

Example

In the standard encoding of $\mathbb{N}$, the length of n of code is $\lceil \log_2 n \rceil$.
In the unary encoding of $\mathbb{N}$, the length of the code of n is n.

The size of the input

After an encoding, we can talk about *the size of the input*: the number of characters in the input character sequence.

Example

In the standard encoding of $\mathbb{N}$, the length of n of code is $\lceil \log_2 n \rceil$. In the unary encoding of $\mathbb{N}$, the length of the code of n is n.

Example

Let G be a simple graph with $V = \{1, 2, \ldots, v\}$ vertex set. To encode it, let $\Sigma = \{0, 1\}$. The length of the code of G will be $\binom{v}{2}$ (the are called *triangular numbers*). The positions of the code are identified by the two-element subsets of V. A character/bit encodes that the corresponding two vertices are connected. Since $2^{\binom{v}{2}}$ is the number of objects to be encoded and $|\Sigma| = 2$ it is not even possible to work with shorter code words to encode all simple graphs with vertices v.

The size of the input II

The size of the input II

Example

Let G be a simple graph with e edges on the vertex set $V = \{1, 2, \ldots, v\}$. Then its code may be to list the elements of V. Each vertex is followed by a colon and the list of its neighbours seperated by semicolons. A vertex and its list of neighbors closed by a period. Hence $\Sigma = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0, ;, .\}$. For example, a coded graph $1 : 2; 3; 4.2 : 1.3 : 1; 5.4 : 1.5 : 3$.

The length of the graph code can be estimated from above $(v + 2e)(\lceil \log_{10} v \rceil + 1)$-gyel. We cannot hope for a more compact encoding in terms of magnitude, since the objects to be encoded are száma $\binom{\binom{v}{2}}{e}$.

Decision problems

Decision problems

Definition: Decision problems

The computational problem f is *decision problem*, if it only takes values $0/1$.

Decision problems

Definition: Decision problems

The computational problem f is *decision problem*, if it only takes values $0/1$.

We can also say (this is the usual way) that an input is either accepted or rejected.

Decision problems

Definition: Decision problems

The computational problem f is *decision problem*, if it only takes values $0/1$.

We can also say (this is the usual way) that an input is either accepted or rejected. So a decision problem can be identified by a subset of Σ^*, the set of inputs to be accepted.

Decision problems

Definition: Decision problems

The computational problem f is *decision problem*, if it only takes values $0/1$.

We can also say (this is the usual way) that an input is either accepted or rejected. So a decision problem can be identified by a subset of Σ^*, the set of inputs to be accepted.

Definition

A subset of words is called a language. That is, a language is $L \subset \Sigma^*$.

Decision problems

Definition: Decision problems

The computational problem f is *decision problem*, if it only takes values $0/1$.

We can also say (this is the usual way) that an input is either accepted or rejected. So a decision problem can be identified by a subset of Σ^*, the set of inputs to be accepted.

Definition

A subset of words is called a language. That is, a language is $L \subset \Sigma^*$.

To summarize: A decision problem can be described by a $L \subset \Sigma^*$.

Decision problems

Definition: Decision problems

The computational problem f is *decision problem*, if it only takes values $0/1$.

We can also say (this is the usual way) that an input is either accepted or rejected. So a decision problem can be identified by a subset of Σ^*, the set of inputs to be accepted.

Definition

A subset of words is called a language. That is, a language is $L \subset \Sigma^*$.

To summarize: A decision problem can be described by a $L \subset \Sigma^*$. Alternatively, the language can be interpreted as a decision problem.

Algorithms: The static picture

Algorithms: The static picture

The algorithms are finitely describable (one could say that they can be coded).

Algorithms: The static picture

The algorithms are finitely describable (one could say that they can be coded).

Such codes can be seen in algorithm theory books or other textbooks, where algorithms are described.

Algorithms: The static picture

The algorithms are finitely describable (one could say that they can be coded).

Such codes can be seen in algorithm theory books or other textbooks, where algorithms are described.

Of course, coding is an agreement among the experts. There are many kinds of possible agreements.

Algorithms: The static picture

The algorithms are finitely describable (one could say that they can be coded).

Such codes can be seen in algorithm theory books or other textbooks, where algorithms are described.

Of course, coding is an agreement among the experts. There are many kinds of possible agreements. Each programming language has a different way of coding algorithms.

Algorithms: Dynamic picture

Algorithms: Dynamic picture

Also there is a dynamic way to look at an algorithm.

Algorithms: Dynamic picture

Also there is a dynamic way to look at an algorithm.

Think of a movie, which shows that for two three-digit numbers, their product is calculated.

Algorithms: Dynamic picture

Also there is a dynamic way to look at an algorithm.

Think of a movie, which shows that for two three-digit numbers, their product is calculated. From time to time, numbers appear on our paper, our workspace is organized and any digit/character has a meaning. If we understand the essence of the claculation we can perform it on different inputs.

Algorithms: Dynamic picture

Also there is a dynamic way to look at an algorithm.

Think of a movie, which shows that for two three-digit numbers, their product is calculated. From time to time, numbers appear on our paper, our workspace is organized and any digit/character has a meaning. If we understand the essence of the claculation we can perform it on different inputs.

Static and dynamic approaches go hand in hand. The two views are equivalent.

Algorithms: Dynamic picture

Also there is a dynamic way to look at an algorithm.

Think of a movie, which shows that for two three-digit numbers, their product is calculated. From time to time, numbers appear on our paper, our workspace is organized and any digit/character has a meaning. If we understand the essence of the claculation we can perform it on different inputs.

Static and dynamic approaches go hand in hand. The two views are equivalent. If the textbook coding (static view) is understood then one is able to execute the algorithm on different inputs.

Algorithms: Dynamic picture

Also there is a dynamic way to look at an algorithm.

Think of a movie, which shows that for two three-digit numbers, their product is calculated. From time to time, numbers appear on our paper, our workspace is organized and any digit/character has a meaning. If we understand the essence of the claculation we can perform it on different inputs.

Static and dynamic approaches go hand in hand. The two views are equivalent. If the textbook coding (static view) is understood then one is able to execute the algorithm on different inputs. The other way around if we saw the execution of an algorithm several times and practiced it well, then we should be able to give some level of description of the of our procedure (assuming some education in programming languages).

Algorithms: Dynamic picture

Also there is a dynamic way to look at an algorithm.

Think of a movie, which shows that for two three-digit numbers, their product is calculated. From time to time, numbers appear on our paper, our workspace is organized and any digit/character has a meaning. If we understand the essence of the claculation we can perform it on different inputs.

Static and dynamic approaches go hand in hand. The two views are equivalent. If the textbook coding (static view) is understood then one is able to execute the algorithm on different inputs. The other way around if we saw the execution of an algorithm several times and practiced it well, then we should be able to give some level of description of the of our procedure (assuming some education in programming languages).

When we define the notion of algorithm we use the dynamic picture.

Algorithms: Dynamic picture

Also there is a dynamic way to look at an algorithm.

Think of a movie, which shows that for two three-digit numbers, their product is calculated. From time to time, numbers appear on our paper, our workspace is organized and any digit/character has a meaning. If we understand the essence of the claculation we can perform it on different inputs.

Static and dynamic approaches go hand in hand. The two views are equivalent. If the textbook coding (static view) is understood then one is able to execute the algorithm on different inputs. The other way around if we saw the execution of an algorithm several times and practiced it well, then we should be able to give some level of description of the of our procedure (assuming some education in programming languages).

When we define the notion of algorithm we use the dynamic picture. At the first encounter with algorithms we also see the dynamic picture.

Historical remarks

Historical remarks

The mathematical concept of an algorithm emerged in the first third of the XX$^{\text{th}}$ century.

Historical remarks

The mathematical concept of an algorithm emerged in the first third of the XX$^{\text{th}}$ century. One of its driving forces was the development of logic, and Hilbert's famous X$^{\text{th}}$ problem.

Historical remarks

The mathematical concept of an algorithm emerged in the first third of the XXth century. One of its driving forces was the development of logic, and Hilbert's famous Xth problem. In it, Hilbert asked whether there is an algorithm that for a given integer polynomial to decide whether it has an integer root (the fundamental problem of Diophantine number theory).

Historical remarks

The mathematical concept of an algorithm emerged in the first third of the XX$^{\text{th}}$ century. One of its driving forces was the development of logic, and Hilbert's famous X$^{\text{th}}$ problem. In it, Hilbert asked whether there is an algorithm that for a given integer polynomial to decide whether it has an integer root (the fundamental problem of Diophantine number theory).

The answer, as it later turned out, was NO.

Historical remarks

The mathematical concept of an algorithm emerged in the first third of the XXth century. One of its driving forces was the development of logic, and Hilbert's famous Xth problem. In it, Hilbert asked whether there is an algorithm that for a given integer polynomial to decide whether it has an integer root (the fundamental problem of Diophantine number theory).

The answer, as it later turned out, was NO. The proof of this is not possible without clarifying the mathematical concept of algorithm.

Historical remarks

The mathematical concept of an algorithm emerged in the first third of the XXth century. One of its driving forces was the development of logic, and Hilbert's famous Xth problem. In it, Hilbert asked whether there is an algorithm that for a given integer polynomial to decide whether it has an integer root (the fundamental problem of Diophantine number theory).

The answer, as it later turned out, was NO. The proof of this is not possible without clarifying the mathematical concept of algorithm.

The word algorithm is part of our everyday vocabulary, but in mathematics it is not are not interpreted.

Historical remarks

The mathematical concept of an algorithm emerged in the first third of the XX$^{\text{th}}$ century. One of its driving forces was the development of logic, and Hilbert's famous X$^{\text{th}}$ problem. In it, Hilbert asked whether there is an algorithm that for a given integer polynomial to decide whether it has an integer root (the fundamental problem of Diophantine number theory).

The answer, as it later turned out, was NO. The proof of this is not possible without clarifying the mathematical concept of algorithm.

The word algorithm is part of our everyday vocabulary, but in mathematics it is not are not interpreted. When one gives a definition the mathematical community must accept it as a correct definition.

Historical remarks

The mathematical concept of an algorithm emerged in the first third of the XXth century. One of its driving forces was the development of logic, and Hilbert's famous Xth problem. In it, Hilbert asked whether there is an algorithm that for a given integer polynomial to decide whether it has an integer root (the fundamental problem of Diophantine number theory).

The answer, as it later turned out, was NO. The proof of this is not possible without clarifying the mathematical concept of algorithm.

The word algorithm is part of our everyday vocabulary, but in mathematics it is not are not interpreted. When one gives a definition the mathematical community must accept it as a correct definition. Church was the first mathematician who was brave enough to propose a definition. It happened on 19th of April 1935, when Church gave a lecture at a meeting of the American Mathematical Society.

Historical remarks (continued)

Historical remarks (continued)

Church said that the quest for the correct definition is over. This statement is referred to as *the Church thesis*.

Historical remarks (continued)

Church said that the quest for the correct definition is over. This statement is referred to as *the Church thesis*.

It is interesting to note that Gödel — who was the unquestioned giant of mathematical logic — did not accept Church's concept of computability.

Historical remarks (continued)

Church said that the quest for the correct definition is over. This statement is referred to as *the Church thesis*.

It is interesting to note that Gödel — who was the unquestioned giant of mathematical logic — did not accept Church's concept of computability.

The real breakthrough was an article by Turing in 1936. In it, he proposed a new definition. It has been widely accepted as the "real" concept of algorithm.

Historical remarks (continued)

Church said that the quest for the correct definition is over. This statement is referred to as *the Church thesis*.

It is interesting to note that Gödel — who was the unquestioned giant of mathematical logic — did not accept Church's concept of computability.

The real breakthrough was an article by Turing in 1936. In it, he proposed a new definition. It has been widely accepted as the "real" concept of algorithm.

Later, the researchers realised that Turing', Church', Gödel' and other's attempts were all equivalent.

Historical remarks (continued)

Church said that the quest for the correct definition is over. This statement is referred to as *the Church thesis*.

It is interesting to note that Gödel — who was the unquestioned giant of mathematical logic — did not accept Church's concept of computability.

The real breakthrough was an article by Turing in 1936. In it, he proposed a new definition. It has been widely accepted as the "real" concept of algorithm.

Later, the researchers realised that Turing', Church', Gödel' and other's attempts were all equivalent.

The thesis is still accepted today.

Break

Configuration of a Turing machine I

Configuration of a Turing machine I

The physical parts of a Turing machine (TM) are: input tape, working tape, output tape, head.

Configuration of a Turing machine I

The physical parts of a Turing machine (TM) are: input tape, working tape, output tape, head.

The tapes contain a sequence of squares/fields/cells. The cells of a t tape are $\{M_i^t\}_{i=0}^{\infty}$ (i.e., M_{100}^{input} is the 100 index/101st cell of the input tape, M_0^{work} is the first/index 0 cell of the work tape).

Configuration of a Turing machine I

The physical parts of a Turing machine (TM) are: input tape, working tape, output tape, head.

The tapes contain a sequence of squares/fields/cells. The cells of a t tape are $\{M_i^t\}_{i=0}^{\infty}$ (i.e., M_{100}^{input} is the 100 index/101st cell of the input tape, M_0^{work} is the first/index 0 cell of the work tape).

The cells should be thought of as a rectangular grid of paper, a row of a square grid, which is infinite in one direction.

Configuration of a Turing machine I

The physical parts of a Turing machine (TM) are: input tape, working tape, output tape, head.

The tapes contain a sequence of squares/fields/cells. The cells of a t tape are $\{M_i^t\}_{i=0}^{\infty}$ (i.e., M_{100}^{input} is the 100 index/101st cell of the input tape, M_0^{work} is the first/index 0 cell of the work tape).

The cells should be thought of as a rectangular grid of paper, a row of a square grid, which is infinite in one direction.

The content of a cell is one character. The cells of the input and output tape contain elements of the Σ alphabet (our computational task is $f : \Sigma \to \Sigma$).

Configuration of a Turing machine I

The physical parts of a Turing machine (TM) are: input tape, working tape, output tape, head.

The tapes contain a sequence of squares/fields/cells. The cells of a t tape are $\{M_i^t\}_{i=0}^{\infty}$ (i.e., M_{100}^{input} is the 100 index/101st cell of the input tape, M_0^{work} is the first/index 0 cell of the work tape).

The cells should be thought of as a rectangular grid of paper, a row of a square grid, which is infinite in one direction.

The content of a cell is one character. The cells of the input and output tape contain elements of the Σ alphabet (our computational task is $f : \Sigma \to \Sigma$).

For the work tape, we use a Γ alphabet, in addition.

Configuration of a TM II

Configuration of a TM II

A Turing machine has a "head"/central processing unit.

Configuration of a TM II

A Turing machine has a "head"/central processing unit.

The head connects with the tapes of the Turing machine as an eye, or hand/pencil. These connections are at certain cell on each tape.

Configuration of a TM II

A Turing machine has a "head"/central processing unit.

The head connects with the tapes of the Turing machine as an eye, or hand/pencil. These connections are at certain cell on each tape.

The "eye" reads the value of a cell, the "hand" can overwrite the content of a cell.

Configuration of a TM II

A Turing machine has a "head"/central processing unit.

The head connects with the tapes of the Turing machine as an eye, or hand/pencil. These connections are at certain cell on each tape.

The "eye" reads the value of a cell, the "hand" can overwrite the content of a cell.

The machine has an input-eye, work-eye/hand and output-hand. The position of these is described by which cell they are located above.

Configuration of a TM II

A Turing machine has a "head"/central processing unit.

The head connects with the tapes of the Turing machine as an eye, or hand/pencil. These connections are at certain cell on each tape.

The "eye" reads the value of a cell, the "hand" can overwrite the content of a cell.

The machine has an input-eye, work-eye/hand and output-hand. The position of these is described by which cell they are located above.

The input tape is a read-only tape, the work tape is a read/write tape, the output tape is a write-only tape.

Configuration of a TM II

A Turing machine has a "head"/central processing unit.

The head connects with the tapes of the Turing machine as an eye, or hand/pencil. These connections are at certain cell on each tape.

The "eye" reads the value of a cell, the "hand" can overwrite the content of a cell.

The machine has an input-eye, work-eye/hand and output-hand. The position of these is described by which cell they are located above.

The input tape is a read-only tape, the work tape is a read/write tape, the output tape is a write-only tape.

The head has a certain state. The possible states form a finite set S.

Configuration of a TM III

Configuration of a TM III

The function of the input tape is to store an input ($\omega \in \Sigma^n$).

Configuration of a TM III

The function of the input tape is to store an input ($\omega \in \Sigma^n$). The work and output tape are one-way infinite tape.

Configuration of a TM III

The function of the input tape is to store an input ($\omega \in \Sigma^n$). The work and output tape are one-way infinite tape.

$\triangleright$ and $\triangleleft$ are very special characters to locate the extreme cells of the tapes.

Configuration of a TM III

The function of the input tape is to store an input ($\omega \in \Sigma^n$). The work and output tape are one-way infinite tape.

$\triangleright$ and $\triangleleft$ are very special characters to locate the extreme cells of the tapes.

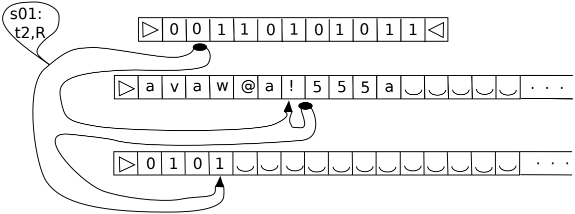
There is a special character used on the work/output tape: $\smile$, the "untouched" character.

Configuration of a TM III

The function of the input tape is to store an input ($\omega \in \Sigma^n$). The work and output tape are one-way infinite tape.

$\triangleright$ and $\triangleleft$ are very special characters to locate the extreme cells of the tapes.

There is a special character used on the work/output tape: $\smile$, the "untouched" character.



Imaginary configuration of an imaginary machine

Configuration of a TM IV

Turing-machine configuration

The configuration in the case of an input of size n is a 7-tuple:

$$\langle \{M_i^{input}\}_{i=0}^{n+1}, \{M_i^{work}\}_{i=0}^{\infty}, \{M_i^{output}\}_{i=0}^{\infty}, p^{input}, p^{work}, p^{output}, s\rangle,$$

where $M_0^{input} = M_0^{work} = M_0^{output} = \triangleright$, $M_{n+1}^{input} = \triangleleft$,
$p^{input} \in \{0, 1, 2, \ldots, n+1\}$, $p^{work}, p^{output} \in \mathbb{N}$, $s \in S$.

The visible part of a configuration

The visible part of a configuration

The head only sees a narrow part of the configuration. This is the content of the cells scanned by the two eyes and its state, i.e. an element of $(\Sigma \cup \{\triangleright, \triangleleft\}) \times (\Gamma \cup \{\triangleright, \smile\}) \times S$.

The visible part of a configuration

The head only sees a narrow part of the configuration. This is the content of the cells scanned by the two eyes and its state, i.e. an element of $(\Sigma \cup \{\triangleright, \triangleleft\}) \times (\Gamma \cup \{\triangleright, \smile\}) \times S$.

title

For a configuration κ $v(\kappa)$ is the visible part of it.

The transition function

The transition function

Based on $v(\kappa)$ the configuration will be updated.

The transition function

Based on $v(\kappa)$ the configuration will be updated.

The movement of the eyes/hands is „continuous”. By this we mean that on each tape the new position differs from the previous one by at most 1.

The transition function

Based on $v(\kappa)$ the configuration will be updated.

The movement of the eyes/hands is „continuous". By this we mean that on each tape the new position differs from the previous one by at most 1.

Definition: Transition function of a Turing machine

The *transition function* of a TM is

$$\delta : (\Sigma \cup \{\triangleright, \triangleleft\}) \times (\Gamma \cup \{\triangleright, \smile\}) \times S \to \{L, ., R\} \times \Gamma \times \{L, ., R\} \times (\{.\} \cup \Sigma) \times S.$$

Hidden assumptions

Hidden assumptions

- i.e. $\delta(\triangleright, \mathit{character}, \mathit{STATE})$ must have a value such that its first coordinate is not L.

Hidden assumptions

- i.e. $\delta(\triangleright, \mathit{character}, \mathit{STATE})$ must have a value such that its first coordinate is not L. The second coordinate is irrelevant because the delimiter cannot be overwritten in the following configuration the contents of the working tape will be the same as before.

Hidden assumptions

- i.e. $\delta(\triangleright, character, STATE)$ must have a value such that its first coordinate is not L. The second coordinate is irrelevant because the delimiter cannot be overwritten in the following configuration the contents of the working tape will be the same as before.
- That is, $\delta(\triangleleft, character, STATE)$ has must have a value such that its first coordinate is not R.

Hidden assumptions

- i.e. $\delta(\triangleright, \mathit{character}, \mathit{STATE})$ must have a value such that its first coordinate is not L. The second coordinate is irrelevant because the delimiter cannot be overwritten in the following configuration the contents of the working tape will be the same as before.
- That is, $\delta(\triangleleft, \mathit{character}, \mathit{STATE})$ has must have a value such that its first coordinate is not R.
- $\delta(\mathit{character}, \triangleright, \mathit{STATE})$ must be such that its third coordinate is not L.

The definition of a Turing machine

The definition of a Turing machine

For a Turing machine we need

The definition of a Turing machine

For a Turing machine we need

(1) a Γ finite alphabet on the work tape,

The definition of a Turing machine

For a Turing machine we need

(1) a Γ finite alphabet on the work tape,

(2) special characters $\triangleright$, $\triangleleft$, $\smile$,

The definition of a Turing machine

For a Turing machine we need

(1) a Γ finite alphabet on the work tape,

(2) special characters $\triangleright$, $\triangleleft$, $\smile$,

(3) a finite set of states S (with special states: START, STOP),

The definition of a Turing machine

For a Turing machine we need

(1) a Γ finite alphabet on the work tape,

(2) special characters $\triangleright$, $\triangleleft$, $\smile$,

(3) a finite set of states S (with special states: START, STOP),

(4) for a δ transition function.

The definition of a Turing machine

For a Turing machine we need

(1) a Γ finite alphabet on the work tape,

(2) special characters $\triangleright$, $\triangleleft$, $\smile$,

(3) a finite set of states S (with special states: START, STOP),

(4) for a δ transition function.

It is important to notice that a specific algorithm/Turing-machine for example uses 2023 states and its working alphabet contains 1001 characters, BUT for each input length it must do "its work".

Subsequent configuration

Subsequent configuration

Let κ be a configuration.

Subsequent configuration

Let κ be a configuration. Easy to take the transition function, evaluate it at $v(\kappa)$,

Subsequent configuration

Let κ be a configuration. Easy to take the transition function, evaluate it at $v(\kappa)$, and update κ according to the value of $\delta(v(\kappa))$.

Subsequent configuration

Let κ be a configuration. Easy to take the transition function, evaluate it at $v(\kappa)$, and update κ according to the value of $\delta(v(\kappa))$.

Definition: Subsequent configuration

The configuration obtained from κ as above κ *is the successor of* κ^+.

Initial configuration of an input ω

Initial configuration of an input ω

Initial configuration of the Turing machine for a given input

Initial configuration for the input $\omega \in \Sigma^n$ for the input of a Turing machine requires some preliminary conventions. Let $START \in S$ be a special state. Let

$$\kappa_0(\omega) = \langle \{M_i^{input}\}_{i=0}^{n+1}, \{M_i^{work}\}_{i=0}^{\infty}, \{M_i^{output}\}_{i=0}^{\infty}, \\ p^{input}, p^{work}, p^{output}, s\rangle,$$

where $M_0^{input} = M_0^{work} = M_0^{output} = \triangleright$, $M_{n+1}^{input} = \triangleleft$, $M_i^{input} = \omega_i$ $(i = 1, 2, \ldots, n)$, $M_i^{output} = M_i^{output} = \smile$ $(i \in \mathbb{N}_+)$, $p^{input} = p^{work} = p^{output} = 0$, $s = START$.

Run of a TM on a given input

Run of a TM on a given input

Definition: Run Turing-machine on given input

Let $\{\kappa_i\}_{i=0}^{\infty}$ be the set of configurations for which $\kappa_0 = \kappa_0(\omega)$ (the initial configuration) and $\kappa_{i+1} = \kappa_i^+$. This sequence of configurations is called *the run associated with the* ω *input*.

Run of a TM on a given input

Definition: Run Turing-machine on given input

Let $\{\kappa_i\}_{i=0}^{\infty}$ be the set of configurations for which $\kappa_0 = \kappa_0(\omega)$ (the initial configuration) and $\kappa_{i+1} = \kappa_i^+$. This sequence of configurations is called *the run associated with the* ω *input*.

The above definition is an infinite configuration sequence is called a run. However, the real algorithm is stops at some point, the computation has completed the calculation, the result is declared.

Truncated run of a TM

Truncated run of a TM

Definition: State STOP

Let $STOP$ be a special state, i.e. $STOP \in S$. Its role is to denote the end of the computation.

Truncated run of a TM

Definition: State STOP

Let $STOP$ be a special state, i.e. $STOP \in S$. Its role is to denote the end of the computation.

An infinite sequence of configurations is *cut off* in some cases.

Truncated run of a TM

Definition: State STOP

Let $STOP$ be a special state, i.e. $STOP \in S$. Its role is to denote the end of the computation.

An infinite sequence of configurations is *cut off* in some cases.

Definition: Reduced/truncated run

Let $\{\kappa_i\}_{i=0}^{\ell}$ be the sequence of configurations for which $\kappa_0 = \kappa_0(\omega)$ (the ω associated with initial configuration) and $\kappa_{i+1} = \kappa_i^+$, and $\ell = \min\{i : \kappa_i \text{ has the state } STOP\}$.

Truncated run of a TM

Definition: State STOP

Let $STOP$ be a special state, i.e. $STOP \in S$. Its role is to denote the end of the computation.

An infinite sequence of configurations is *cut off* in some cases.

Definition: Reduced/truncated run

Let $\{\kappa_i\}_{i=0}^{\ell}$ be the sequence of configurations for which $\kappa_0 = \kappa_0(\omega)$ (the ω associated with initial configuration) and $\kappa_{i+1} = \kappa_i^+$, and $\ell = \min\{i : \kappa_i \text{ has the state } STOP\}$.

Note that $\min \emptyset = \infty$.

Truncated run of a TM

Definition: State STOP

Let $STOP$ be a special state, i.e. $STOP \in S$. Its role is to denote the end of the computation.

An infinite sequence of configurations is *cut off* in some cases.

Definition: Reduced/truncated run

Let $\{\kappa_i\}_{i=0}^{\ell}$ be the sequence of configurations for which $\kappa_0 = \kappa_0(\omega)$ (the ω associated with initial configuration) and $\kappa_{i+1} = \kappa_i^+$, and $\ell = \min\{i : \kappa_i \text{ has the state } STOP\}$.

Note that $\min \emptyset = \infty$.

If $\ell < \infty$, then we say that the run is finite.

Truncated run of a TM

Definition: State STOP

Let $STOP$ be a special state, i.e. $STOP \in S$. Its role is to denote the end of the computation.

An infinite sequence of configurations is *cut off* in some cases.

Definition: Reduced/truncated run

Let $\{\kappa_i\}_{i=0}^{\ell}$ be the sequence of configurations for which $\kappa_0 = \kappa_0(\omega)$ (the ω associated with initial configuration) and $\kappa_{i+1} = \kappa_i^+$, and $\ell = \min\{i : \kappa_i \text{ has the state } STOP\}$.

Note that $\min \emptyset = \infty$.

If $\ell < \infty$, then we say that the run is finite. We say that the machine stops on input ω.

Truncated run of a TM

Definition: State STOP

Let $STOP$ be a special state, i.e. $STOP \in S$. Its role is to denote the end of the computation.

An infinite sequence of configurations is *cut off* in some cases.

Definition: Reduced/truncated run

Let $\{\kappa_i\}_{i=0}^{\ell}$ be the sequence of configurations for which $\kappa_0 = \kappa_0(\omega)$ (the ω associated with initial configuration) and $\kappa_{i+1} = \kappa_i^+$, and $\ell = \min\{i : \kappa_i$ has the state $STOP\}$.

Note that $\min \emptyset = \infty$.

If $\ell < \infty$, then we say that the run is finite. We say that the machine stops on input ω.

If on ω the machine stops then the *computed output* is the content of the output tape (ignoring $\triangleright$ and $\smile$'s).

The function calculated by a TM

The function calculated by a TM

Definition

A function f is computed by a Turing machine T if for each $\omega \in \Sigma^*$ the Turing machine stops and the computed value is $f(\omega)$.

The function calculated by a TM

Definition

A function f is computed by a Turing machine T if for each $\omega \in \Sigma^*$ the Turing machine stops and the computed value is $f(\omega)$.

Another way of putting it is.

The function calculated by a TM

Definition

A function f is computed by a Turing machine T if for each $\omega \in \Sigma^*$ the Turing machine stops and the computed value is $f(\omega)$.

Another way of putting it is.

Definition

Every T Turing machine computes a $f_T : \Sigma^* \to \Sigma^* \cup \{\infty\}$ function. For $\omega \in \Sigma^*$, $f_T(\omega) = \infty$, if the run is not finite, and $f_T(\omega)$ is the calculated element of Σ^* if the run is finite at ω.

The function calculated by a TM

Definition

A function f is computed by a Turing machine T if for each $\omega \in \Sigma^*$ the Turing machine stops and the computed value is $f(\omega)$.

Another way of putting it is.

Definition

Every T Turing machine computes a $f_T : \Sigma^* \to \Sigma^* \cup \{\infty\}$ function. For $\omega \in \Sigma^*$, $f_T(\omega) = \infty$, if the run is not finite, and $f_T(\omega)$ is the calculated element of Σ^* if the run is finite at ω.

T computes a function f if $f = f_T$.

Computable functions

Computable functions

Definition

A function $f : \Sigma^* \to \Sigma^*$ is *computable* if for a suitable T Turing machine $f = f_T$.

Computable functions

Definition

A function $f : \Sigma^* \to \Sigma^*$ is *computable* if for a suitable T Turing machine $f = f_T$.

For a computable function, many suitable T Turing machines/algorithms exist.

Computable functions

Definition

A function $f : \Sigma^* \to \Sigma^*$ is *computable* if for a suitable T Turing machine $f = f_T$.

For a computable function, many suitable T Turing machines/algorithms exist.

We note in advance that there are non-computable functions.

Decider TM

Decider TM

A Turing machine, that solves a decision problem the output tape is used only to write a single bit.

Decider TM

A Turing machine, that solves a decision problem the output tape is used only to write a single bit.

The definition can be simplified.

Decider TM

A Turing machine, that solves a decision problem the output tape is used only to write a single bit.

The definition can be simplified. The output tape is discardable.

Decider TM

A Turing machine, that solves a decision problem the output tape is used only to write a single bit.

The definition can be simplified. The output tape is discardable. Replace the STOP state with ACCEPT and REJECT states.

Definition

For decision tasks we use the above model, this *decider Turing machine*.

Decider TM

A Turing machine, that solves a decision problem the output tape is used only to write a single bit.

The definition can be simplified. The output tape is discardable. Replace the STOP state with ACCEPT and REJECT states.

Definition

For decision tasks we use the above model, this *decider Turing machine*.

Then the run stops if and only if, if it reaches the REJECT or ACCEPT state.

Decider TM

A Turing machine, that solves a decision problem the output tape is used only to write a single bit.

The definition can be simplified. The output tape is discardable. Replace the STOP state with ACCEPT and REJECT states.

Definition

For decision tasks we use the above model, this *decider Turing machine*.

Then the run stops if and only if, if it reaches the REJECT or ACCEPT state.

Definition

A decider Turing machine, T decides the language L, if for all $\omega \in L$, the machine stops with ACCEPT and for all $\omega \notin L$ the machine terminates with REJECT. (Specifically, the machine stops on all inputs.)

Decidable languages

Decidable languages

Definition

A $L \subset \Sigma^*$ language is decidable if there is a Turing machine that "solves" it.

Decidable languages

Definition

A $L \subset \Sigma^*$ language is decidable if there is a Turing machine that "solves" it.

The set of decidable languages is denoted by $\mathcal{D}(\Sigma)$.

Enumerable languages

Enumerable languages

We mention an additional option for acceptance/rejection.

Enumerable languages

We mention an additional option for acceptance/rejection.

Definition

An enumerating Turing machine T *defines* the language L, if for all $\omega \in L$, the machine stops with ACCEPT status and for all $\omega \notin L$ it doesn't stop (loops forever).

Enumerable languages

We mention an additional option for acceptance/rejection.

Definition

An enumerating Turing machine T *defines* the language L, if for all $\omega \in L$, the machine stops with ACCEPT status and for all $\omega \notin L$ it doesn't stop (loops forever).

Definition

The set of enumerable languages is be $\mathcal{S}(\Sigma)$.

Enumerable languages

We mention an additional option for acceptance/rejection.

Definition

An enumerating Turing machine T *defines* the language L, if for all $\omega \in L$, the machine stops with ACCEPT status and for all $\omega \notin L$ it doesn't stop (loops forever).

Definition

The set of enumerable languages is be $\mathcal{S}(\Sigma)$.

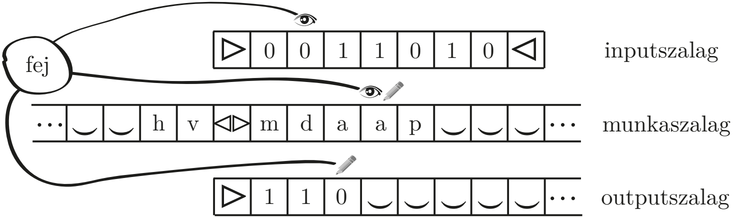
For a given alphabet Σ, obviously

$$\mathcal{D}(\Sigma) \subset \mathcal{S}(\Sigma) \subset \mathcal{P}(\Sigma^*).$$

Break

Variations: two-way infinite work tape

Variations: two-way infinite work tape



The two-way infinite working tape

.

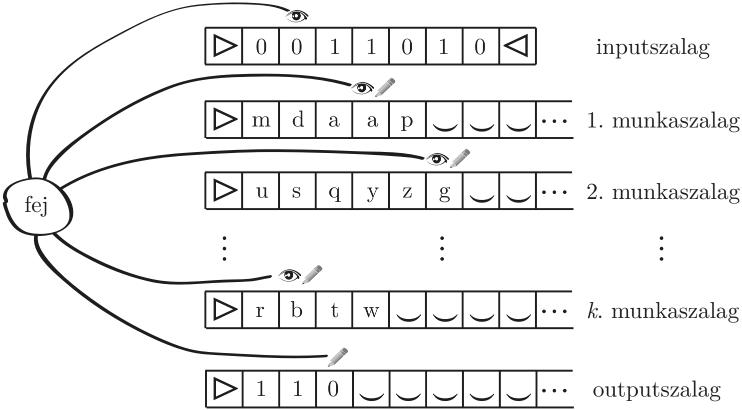
Variations: k-tape model

Variations: k-tape model

$k \in \mathbb{N}$ is fixed.

Variations: k-tape model

$k \in \mathbb{N}$ is fixed.



The k-tape model

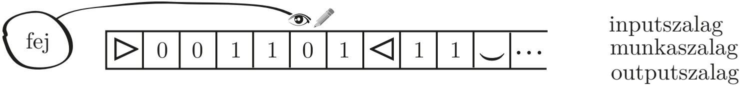
Variations: single-tape model

Variations: single-tape model

All operations on a single strip which is therefore an input, a work, and an output tape at the same time. A tape is right infinite and the initial fields are occupied by the input $\triangleright$,$\triangleleft$ between signals. In this model, the input can also be overwritten and the output is the contents of the tape when the $STOP \in S$ state state is reached.

Variations: single-tape model

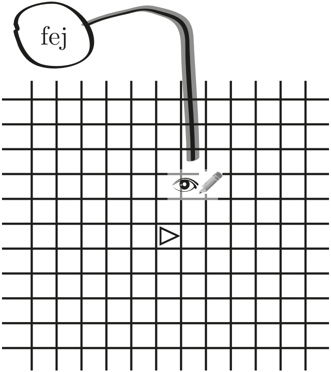
All operations on a single strip which is therefore an input, a work, and an output tape at the same time. A tape is right infinite and the initial fields are occupied by the input $\triangleright, \triangleleft$ between signals. In this model, the input can also be overwritten and the output is the contents of the tape when the $STOP \in S$ state state is reached.



The single-tape model

Variant: two-dimensional work space

Variant: two-dimensional work space



2-dimensional work space

Robustness

Robustness

Theorem

$\mathcal{D}(\Sigma) \subset \mathcal{S}(\Sigma)$ doesn't depend on the model.

Default model

If we say that a L language is decidable, we mean that there is a suitable $k \in \mathbb{N}$ and a suitable k-tape Turing machine which decides L.

This is the end!

Thank you for your attention!