Complexity Theory

Turing Machines and Languages

Markus Krötzsch

TU Dresden

11 Oct 2017

CC BY SA

https://iccl.inf.tu-dresden.de/web/Complexity_Theory_(WS2017/18)/en

Deterministic Turing Machines

A Model for Computation

Clear

To understand computational problems we need to have a formal understanding of what an **algorithm** is.

A Model for Computation

Clear

To understand computational problems we need to have a formal understanding of what an **algorithm** is.

Example 2.1 (Hilbert's Tenth Problem)

> "Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers." (→ Wikipedia)

A Model for Computation

Clear

To understand computational problems we need to have a formal understanding of what an **algorithm** is.

Example 2.1 (Hilbert's Tenth Problem)

> "Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers." (→ Wikipedia)

Question

How can we model the notion of an algorithm?

A Model for Computation

Clear

To understand computational problems we need to have a formal understanding of what an **algorithm** is.

Example 2.1 (Hilbert's Tenth Problem)

> "Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers." (→ Wikipedia)

Question

How can we model the notion of an algorithm?

Answer

With Turing machines.

Turing Machines

Let us fix a blank symbol $\square$.

Turing Machines

Let us fix a blank symbol $\square$.

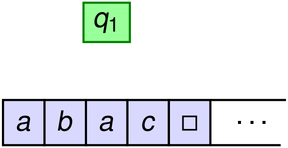
Definition 2.2

A (deterministic) **Turing Machine** $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ consists of

- a finite set Q of **states**,
- an **input alphabet** Σ not containing $\square$,
- a **tape alphabet** Γ such that $\Gamma \supseteq \Sigma \cup \{\square\}$.
- a **transition function** $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{\mathsf{L}, \mathsf{R}\}$
- an **initial state** $q_0 \in Q$,
- an **accepting state** $q_{\text{accept}} \in Q$, and
- an **rejecting state** $q_{\text{reject}} \in Q$ such that $q_{\text{accept}} \neq q_{\text{reject}}$.

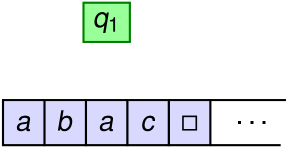
Turing Machines

Example 2.3



Turing Machines

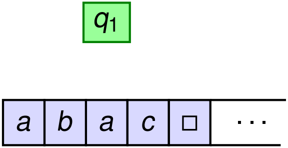
Example 2.3



- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over Γ, followed by an infinite sequence of $\square$.

Turing Machines

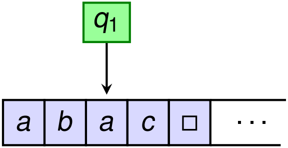
Example 2.3



- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over Γ, followed by an infinite sequence of □.
- The head of the machine is at exactly one position of the tape

Turing Machines

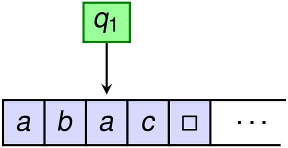
Example 2.3



- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over Γ, followed by an infinite sequence of $\square$.
- The head of the machine is at exactly one position of the tape

Turing Machines

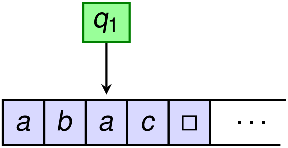
Example 2.3



- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over Γ, followed by an infinite sequence of $\square$.
- The head of the machine is at exactly one position of the tape
- The head can read only one symbol at a time

Turing Machines

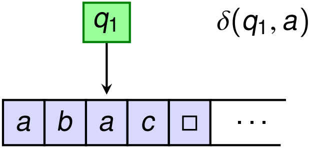
Example 2.3



- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over Γ, followed by an infinite sequence of $\square$.
- The head of the machine is at exactly one position of the tape
- The head can read only one symbol at a time
- The head moves and writes according to the transition function δ; the current state also changes accordingly

Turing Machines

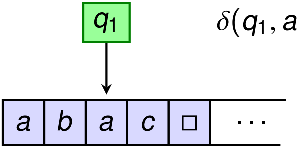
Example 2.3



- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over Γ, followed by an infinite sequence of $\square$.
- The head of the machine is at exactly one position of the tape
- The head can read only one symbol at a time
- The head moves and writes according to the transition function δ; the current state also changes accordingly

Turing Machines

Example 2.3

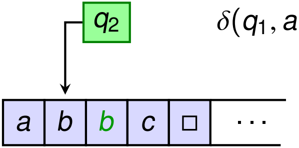


$$\delta(q_1, a) = (q_2, b, \mathsf{L})$$

- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over Γ, followed by an infinite sequence of $\square$.
- The head of the machine is at exactly one position of the tape
- The head can read only one symbol at a time
- The head moves and writes according to the transition function δ; the current state also changes accordingly

Turing Machines

Example 2.3

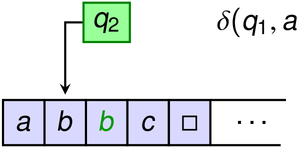


$$\delta(q_1, a) = (q_2, b, \mathsf{L})$$

- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over Γ, followed by an infinite sequence of □.
- The head of the machine is at exactly one position of the tape
- The head can read only one symbol at a time
- The head moves and writes according to the transition function δ; the current state also changes accordingly

Turing Machines

Example 2.3



$$\delta(q_1, a) = (q_2, b, \mathsf{L})$$

- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over Γ, followed by an infinite sequence of □.
- The head of the machine is at exactly one position of the tape
- The head can read only one symbol at a time
- The head moves and writes according to the transition function δ; the current state also changes accordingly
- The head will stay put when attempting to cross the left tape end

Configurations

Observation: to describe the current step of a computation of a TM it is enough to know

- the content of the tape,
- the current state, and
- the position of the head

Configurations

Observation: to describe the current step of a computation of a TM it is enough to know

- the content of the tape,
- the current state, and
- the position of the head

Definition 2.4

A **configuration** of a TM $\mathcal{M}$ is a word *uqv* such that

- $q \in Q$,
- $uv \in \Gamma^*$

Configurations

Observation: to describe the current step of a computation of a TM it is enough to know

- the content of the tape,
- the current state, and
- the position of the head

Definition 2.4

A **configuration** of a TM $\mathcal{M}$ is a word uqv such that

- $q \in Q$,
- $uv \in \Gamma^*$

Some special configurations:

- The **start configuration** for some input word $w \in \Sigma^*$ is the configuration q_0w
- A configuration uqv is **accepting** if $q = q_{\text{accept}}$.
- A configuration uqv is **rejecting** if $q = q_{\text{reject}}$.

Computation

We write

- $\mathcal{C} \vdash_{\mathcal{M}} \mathcal{C}'$ only if $\mathcal{C}'$ can be reached from $\mathcal{C}$ by one computation step of $\mathcal{M}$;
- $\mathcal{C} \vdash^{*}_{\mathcal{M}} \mathcal{C}'$ only if $\mathcal{C}'$ can be reached from $\mathcal{C}$ in a finite number of computation steps of $\mathcal{M}$.

Computation

We write

- $\mathcal{C} \vdash_{\mathcal{M}} \mathcal{C}'$ only if $\mathcal{C}'$ can be reached from $\mathcal{C}$ by one computation step of $\mathcal{M}$;
- $\mathcal{C} \vdash_{\mathcal{M}}^{*} \mathcal{C}'$ only if $\mathcal{C}'$ can be reached from $\mathcal{C}$ in a finite number of computation steps of $\mathcal{M}$.

We say that $\mathcal{M}$ **halts** on input w if and only if there is a finite sequence of configurations

$$\mathcal{C}_0 \vdash_{\mathcal{M}} \mathcal{C}_1 \vdash_{\mathcal{M}} \cdots \vdash_{\mathcal{M}} \mathcal{C}_\ell$$

such that $\mathcal{C}_0$ is the start configuration of $\mathcal{M}$ on input w and $\mathcal{C}_\ell$ is an accepting or rejecting configuration. Otherwise $\mathcal{M}$ **loops** on input w.

Computation

We write

- $\mathcal{C} \vdash_{\mathcal{M}} \mathcal{C}'$ only if $\mathcal{C}'$ can be reached from $\mathcal{C}$ by one computation step of $\mathcal{M}$;
- $\mathcal{C} \vdash^{*}_{\mathcal{M}} \mathcal{C}'$ only if $\mathcal{C}'$ can be reached from $\mathcal{C}$ in a finite number of computation steps of $\mathcal{M}$.

We say that $\mathcal{M}$ **halts** on input w if and only if there is a finite sequence of configurations

$$\mathcal{C}_0 \vdash_{\mathcal{M}} \mathcal{C}_1 \vdash_{\mathcal{M}} \cdots \vdash_{\mathcal{M}} \mathcal{C}_\ell$$

such that $\mathcal{C}_0$ is the start configuration of $\mathcal{M}$ on input w and $\mathcal{C}_\ell$ is an accepting or rejecting configuration. Otherwise $\mathcal{M}$ **loops** on input w.

We say that $\mathcal{M}$ **accepts** the input w only if $\mathcal{M}$ halts on input w with an accepting configuration.

Recognisability and Decidability

Recognisability and Decidability

Definition 2.5

Let $\mathcal{M}$ be a Turing machine with input alphabet Σ. The **language accepted by $\mathcal{M}$** is the set

$$\mathcal{L}(\mathcal{M}) := \{ w \in \Sigma^* \mid \mathcal{M} \text{ accepts } w \}.$$

Recognisability and Decidability

Definition 2.5

Let $\mathcal{M}$ be a Turing machine with input alphabet Σ. The **language accepted by $\mathcal{M}$** is the set

$$\mathcal{L}(\mathcal{M}) := \{ w \in \Sigma^* \mid \mathcal{M} \text{ accepts } w \}.$$

A language $\mathcal{L} \subseteq \Sigma^*$ is called **Turing-recognisable** (**recursively enumerable**) if and only if there exists a Turing machine $\mathcal{M}$ with input alphabet Σ^* such that $\mathcal{L} = \mathcal{L}(\mathcal{M})$.
In this case we say that $\mathcal{M}$ **recognises** $\mathcal{L}$.

Recognisability and Decidability

Definition 2.5

Let $\mathcal{M}$ be a Turing machine with input alphabet Σ. The **language accepted by $\mathcal{M}$** is the set

$$\mathcal{L}(\mathcal{M}) := \{ w \in \Sigma^* \mid \mathcal{M} \text{ accepts } w \}.$$

A language $\mathcal{L} \subseteq \Sigma^*$ is called **Turing-recognisable** (**recursively enumerable**) if and only if there exists a Turing machine $\mathcal{M}$ with input alphabet Σ^* such that $\mathcal{L} = \mathcal{L}(\mathcal{M})$.
In this case we say that $\mathcal{M}$ **recognises** $\mathcal{L}$.

A language $\mathcal{L} \subseteq \Sigma^*$ is called **Turing-decidable** (**decidable**, **recursive**) if and only if there exists a Turing machine $\mathcal{M}$ such that $\mathcal{L} = \mathcal{L}(\mathcal{M})$ and $\mathcal{M}$ halts on every input. In this case we say that $\mathcal{M}$ **decides** $\mathcal{L}$.

Example

Claim

The language $\mathcal{L} := \{ \mathrm{a}^{2^n} \mid n \geq 0 \}$ is decidable.

Example

Claim

The language $\mathcal{L} := \{ \mathrm{a}^{2^n} \mid n \geq 0 \}$ is decidable.

Proof

A Turing machine $\mathcal{M}$ that decides $\mathcal{L}$ is

$\mathcal{M}$:= On input w, where w is a string

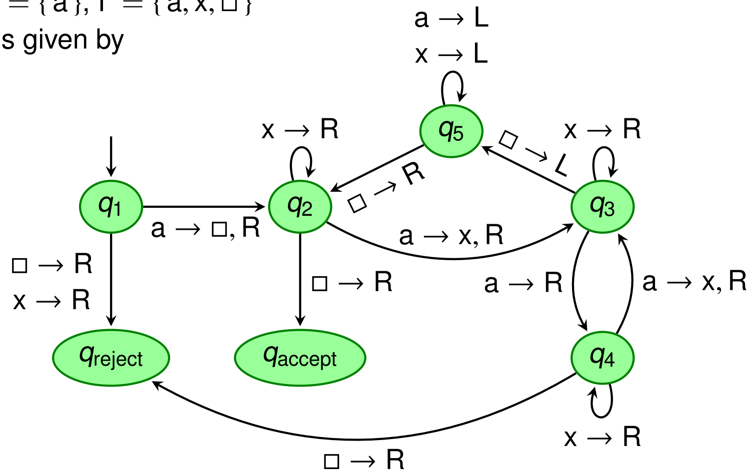
- Go from left to right over the tape and cross off every other 0
- If in the first step the tape contained a single 0, *accept*
- If in the first step the number of 0s on the tape was odd, *reject*
- Return the head the beginning of the tape
- Go to the first step

Example (cont'd)

Formally, $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$, where

- $Q = \{ q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}} \}$
- $\Sigma = \{ \text{a} \}$, $\Gamma = \{ \text{a}, \text{x}, \square \}$

and δ is given by



Problems as Languages

Observation

- Languages can be used to model computational problems.
- For this, a suitable **encoding** is necessary
- TMs must be able to decode the encoding

Problems as Languages

Observation

- Languages can be used to model computational problems.
- For this, a suitable **encoding** is necessary
- TMs must be able to decode the encoding

Example 2.6 (Graph-Connectedness)

The question whether a graph is connected or not can be seen as the **word problem** of the following language

$$\text{GCONN} := \{ \langle G \rangle \mid G \text{ is a connected graph} \},$$

where $\langle G \rangle$ is (for example) the adjacency matrix encoded in binary.

Problems as Languages

Observation

- Languages can be used to model computational problems.
- For this, a suitable **encoding** is necessary
- TMs must be able to decode the encoding

Example 2.6 (Graph-Connectedness)

The question whether a graph is connected or not can be seen as the **word problem** of the following language

$$\text{GCONN} := \{ \langle G \rangle \mid G \text{ is a connected graph} \},$$

where $\langle G \rangle$ is (for example) the adjacency matrix encoded in binary.

Notation

The encoding of objects $\mathcal{O}_1, \ldots, \mathcal{O}_n$ we denote by $\langle \mathcal{O}_1, \ldots, \mathcal{O}_n \rangle$.

The Church-Turing Thesis

It turns out that Turing-machines are **equivalent** to a number of formalisations of the intuitive notion of an **algorithm**

- λ-calculus
- while-programs
- μ-recursive functions
- Random-Access Machines
- ...

The Church-Turing Thesis

It turns out that Turing-machines are **equivalent** to a number of formalisations of the intuitive notion of an **algorithm**

- λ-calculus
- while-programs
- μ-recursive functions
- Random-Access Machines
- ...

Because of this it is believed that Turing-machines completely capture the intuitive notion of an algorithm. $\leadsto$ **Church-Turing Thesis**:

> *"A function on the natural numbers is intuitively computable if and only if it can be computed by a Turing machine."*

($\rightarrow$ Wikipedia: Church-Turing Thesis)

Variants of Turing Machines

Variations of Turing-Machines

It has also been shown that deterministic, single-tape Turing machines are equivalent to a wide range of other forms of Turing machines:

- Multi-tape Turing machines
- Nondeterministic Turing machines
- Turing machines with doubly-infinite tape
- Multi-head Turing machines
- Two-dimensional Turing machines
- Write-once Turing machines
- Two-stack machines
- Two-counter machines
- ...

Variations of Turing-Machines

It has also been shown that deterministic, single-tape Turing machines are equivalent to a wide range of other forms of Turing machines:

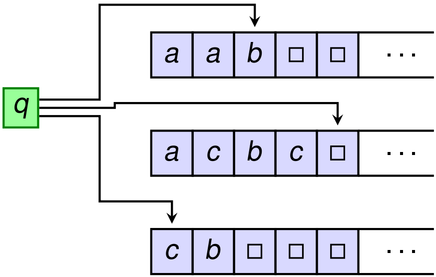
- Multi-tape Turing machines
- Nondeterministic Turing machines
- Turing machines with doubly-infinite tape
- Multi-head Turing machines
- Two-dimensional Turing machines
- Write-once Turing machines
- Two-stack machines
- Two-counter machines
- …

Multi-Tape Turing Machines

k-tape Turing machines are a variant of Turing machines that have k tapes.

Multi-Tape Turing Machines

***k*-tape Turing machines** are a variant of Turing machines that have k tapes.



Multi-Tape Turing Machines

Definition 2.7

Let $k \in \mathbb{N}$. Then a (deterministic) ***k*-tape Turing machine** is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where

- Q, Σ, Γ, q_0, q_{accept}, q_{reject} are as for TMs
- δ is a transition function for k tapes, i.e.,

$$\delta\colon Q \times \Gamma^k \to Q \times \Gamma^k \times \{\mathsf{L}, \mathsf{R}, \mathsf{N}\}^k$$

Multi-Tape Turing Machines

Definition 2.7

Let $k \in \mathbb{N}$. Then a (deterministic) ***k*-tape Turing machine** is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where

- Q, Σ, Γ, q_0, q_{accept}, q_{reject} are as for TMs
- δ is a transition function for k tapes, i.e.,

$$\delta \colon Q \times \Gamma^k \to Q \times \Gamma^k \times \{\,\mathsf{L}, \mathsf{R}, \mathsf{N}\,\}^k$$

Running M on input $w \in \Sigma^*$ means to start M with the content of the first tape being w and all other tapes blank.

Multi-Tape Turing Machines

Definition 2.7

Let $k \in \mathbb{N}$. Then a (deterministic) **k-tape Turing machine** is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where

- Q, Σ, Γ, q_0, q_{accept}, q_{reject} are as for TMs
- δ is a transition function for k tapes, i.e.,

$$\delta\colon Q \times \Gamma^k \to Q \times \Gamma^k \times \{\,\mathsf{L}, \mathsf{R}, \mathsf{N}\,\}^k$$

Running M on input $w \in \Sigma^*$ means to start M with the content of the first tape being w and all other tapes blank.

The notions of a **configuration** and of the **language accepted by M** are defined analogously to the single-tape case.

Multi-Tape Turing Machines

Theorem 2.8

Every multi-tape Turing machine has an equivalent single-tape Turing machine.

Multi-Tape Turing Machines

Theorem 2.8

Every multi-tape Turing machine has an equivalent single-tape Turing machine.

Proof.

Let M be a k-tape Turing machine. Simulate M with a single-tape TM S by

Multi-Tape Turing Machines

Theorem 2.8

Every multi-tape Turing machine has an equivalent single-tape Turing machine.

Proof.

Let M be a k-tape Turing machine. Simulate M with a single-tape TM S by

- keeping the content of all k tapes on a single tape, separated by #

Multi-Tape Turing Machines

Theorem 2.8

Every multi-tape Turing machine has an equivalent single-tape Turing machine.

Proof.

Let M be a k-tape Turing machine. Simulate M with a single-tape TM S by

- keeping the content of all k tapes on a single tape, separated by #
- marking the positions of the individual heads using special symbols

Multi-Tape Turing Machines

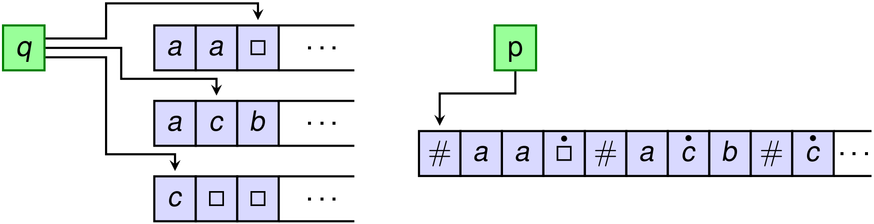
Theorem 2.8

Every multi-tape Turing machine has an equivalent single-tape Turing machine.

Proof.

Let M be a k-tape Turing machine. Simulate M with a single-tape TM S by

- keeping the content of all k tapes on a single tape, separated by #
- marking the positions of the individual heads using special symbols



Multi-Tape Turing Machines

$S :=$ On input $w = w_1 \dots w_n$

Multi-Tape Turing Machines

$S :=$ On input $w = w_1 \ldots w_n$

- Format the tape to contain the word

$$\#\dot{w}_1 w_2 \ldots w_n \# \dot{\square} \# \dot{\square} \# \ldots \#$$

Multi-Tape Turing Machines

$S :=$ On input $w = w_1 \ldots w_n$

- Format the tape to contain the word

$$\#\dot{w}_1 w_2 \ldots w_n \# \dot{\square} \# \dot{\square} \# \ldots \#$$

- Scan the tape from the first # to the $(k+1)$-th # to determine the symbols below the markers.

Multi-Tape Turing Machines

$S :=$ On input $w = w_1 \dots w_n$

- Format the tape to contain the word

$$\#\dot{w}_1 w_2 \dots w_n \# \dot{\square} \# \dot{\square} \# \dots \#$$

- Scan the tape from the first # to the $(k+1)$-th # to determine the symbols below the markers.
- Update all tapes according to M's transition function with a second pass over the tape; if any head of M moves to some previously unread portion of its tape, insert a blank symbol at the corresponding position and shift the right tape contents by one cell

Multi-Tape Turing Machines

$S :=$ On input $w = w_1 \ldots w_n$

- Format the tape to contain the word

$$\#\dot{w}_1 w_2 \ldots w_n \# \dot{\square} \# \dot{\square} \# \ldots \#$$

- Scan the tape from the first # to the $(k+1)$-th # to determine the symbols below the markers.
- Update all tapes according to M's transition function with a second pass over the tape; if any head of M moves to some previously unread portion of its tape, insert a blank symbol at the corresponding position and shift the right tape contents by one cell
- Repeat until the accepting or rejection state is reached.

Multi-Tape Turing Machines

$S :=$ On input $w = w_1 \ldots w_n$

- Format the tape to contain the word

$$\#\dot{w}_1 w_2 \ldots w_n \# \dot{\square} \# \dot{\square} \# \ldots \#$$

- Scan the tape from the first # to the $(k+1)$-th # to determine the symbols below the markers.
- Update all tapes according to M's transition function with a second pass over the tape; if any head of M moves to some previously unread portion of its tape, insert a blank symbol at the corresponding position and shift the right tape contents by one cell
- Repeat until the accepting or rejection state is reached.

□

Nondeterministic Turing Machines

Goal

Allow transitions to be **nondeterministic**.

Nondeterministic Turing Machines

Goal

Allow transitions to be **nondeterministic**.

Approach

Change transition function from

$$\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{\, \mathsf{L}, \mathsf{R} \,\}$$

to

$$\delta\colon Q \times \Gamma \to 2^{Q \times \Gamma \times \{\, \mathsf{L}, \mathsf{R} \,\}}.$$

Nondeterministic Turing Machines

Goal

Allow transitions to be **nondeterministic**.

Approach

Change transition function from

$$\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{\, \mathsf{L}, \mathsf{R} \,\}$$

to

$$\delta\colon Q \times \Gamma \to 2^{Q \times \Gamma \times \{\, \mathsf{L}, \mathsf{R} \,\}}.$$

The notions of **accepting** and **rejecting computations** are defined accordingly. Note: there may be more than one or no computation of a nondeterministic TM on a given input.

Nondeterministic Turing Machines

Goal

Allow transitions to be **nondeterministic**.

Approach

Change transition function from

$$\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{\, \mathsf{L}, \mathsf{R} \,\}$$

to

$$\delta\colon Q \times \Gamma \to 2^{Q \times \Gamma \times \{\, \mathsf{L}, \mathsf{R} \,\}}.$$

The notions of **accepting** and **rejecting computations** are defined accordingly. Note: there may be more than one or no computation of a nondeterministic TM on a given input.
A nondeterministic TM M **accepts** an input w if and only if **there exists** some accepting computation of M on input w.

Nondeterministic Turing Machines

Theorem 2.9

Every nondeterministic TM has an equivalent deterministic TM.

Nondeterministic Turing Machines

Theorem 2.9

Every nondeterministic TM has an equivalent deterministic TM.

Proof.

Let N be a nondeterministic TM. We construct a deterministic TM D that is equivalent to N, i.e., $\mathcal{L}(N) = \mathcal{L}(D)$.

Nondeterministic Turing Machines

Theorem 2.9

Every nondeterministic TM has an equivalent deterministic TM.

Proof.

Let N be a nondeterministic TM. We construct a deterministic TM D that is equivalent to N, i.e., $\mathcal{L}(N) = \mathcal{L}(D)$.

Idea

Nondeterministic Turing Machines

Theorem 2.9

Every nondeterministic TM has an equivalent deterministic TM.

Proof.

Let N be a nondeterministic TM. We construct a deterministic TM D that is equivalent to N, i.e., $\mathcal{L}(N) = \mathcal{L}(D)$.

Idea

- D deterministically traverses in breath-first order the tree of configuration of N, where each branch represents a different possibility for N to continue.

Nondeterministic Turing Machines

Theorem 2.9

Every nondeterministic TM has an equivalent deterministic TM.

Proof.

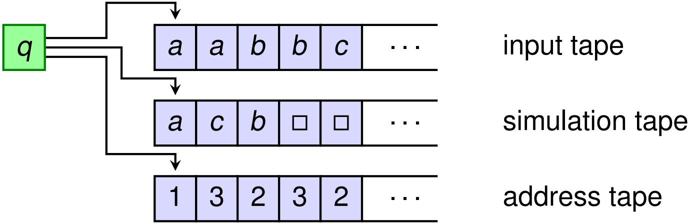
Let N be a nondeterministic TM. We construct a deterministic TM D that is equivalent to N, i.e., $\mathcal{L}(N) = \mathcal{L}(D)$.

Idea

- D deterministically traverses in breath-first order the tree of configuration of N, where each branch represents a different possibility for N to continue.
- For this, successively try out all possible choices of transitions allowed by N.

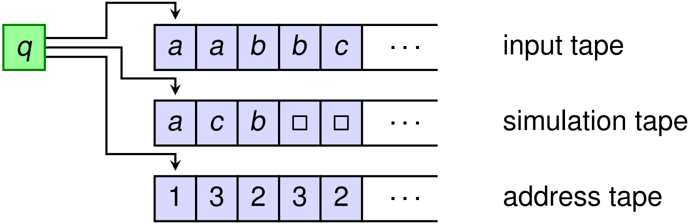
Nondeterministic Turing Machines

Sketch of D:



Nondeterministic Turing Machines

Sketch of D:



Let b be the maximal number of choices in δ, i.e.,

$$b := \max\left\{ |\delta(q,x)| \;\middle|\; q \in Q, x \in \Gamma \right\}.$$

Nondeterministic Turing Machines

D works as follows:

Nondeterministic Turing Machines

D works as follows:

(1) Start: input tape contains input *w*, simulation and address tape empty

Nondeterministic Turing Machines

D works as follows:

(1) Start: input tape contains input *w*, simulation and address tape empty
(2) Copy *w* to the simulation tape and initialize the address tape with 0.

Nondeterministic Turing Machines

D works as follows:

(1) Start: input tape contains input *w*, simulation and address tape empty

(2) Copy *w* to the simulation tape and initialize the address tape with 0.

(3) Simulate one finite computation of *N* on *w* on the simulation tape.

Nondeterministic Turing Machines

D works as follows:

(1) Start: input tape contains input *w*, simulation and address tape empty
(2) Copy *w* to the simulation tape and initialize the address tape with 0.
(3) Simulate one finite computation of *N* on *w* on the simulation tape.
    - Interpret the address tape as a list of choices to make during this computation.

Nondeterministic Turing Machines

D works as follows:

(1) Start: input tape contains input *w*, simulation and address tape empty

(2) Copy *w* to the simulation tape and initialize the address tape with 0.

(3) Simulate one finite computation of *N* on *w* on the simulation tape.
   - Interpret the address tape as a list of choices to make during this computation.
   - If a choice is invalid, abort simulation.

Nondeterministic Turing Machines

D works as follows:

(1) Start: input tape contains input *w*, simulation and address tape empty

(2) Copy *w* to the simulation tape and initialize the address tape with 0.

(3) Simulate one finite computation of *N* on *w* on the simulation tape.

- Interpret the address tape as a list of choices to make during this computation.
- If a choice is invalid, abort simulation.
- If an accepting configuration is reached at the end of the simulation, *accept*.

Nondeterministic Turing Machines

D works as follows:

(1) Start: input tape contains input *w*, simulation and address tape empty
(2) Copy *w* to the simulation tape and initialize the address tape with 0.
(3) Simulate one finite computation of *N* on *w* on the simulation tape.
  - Interpret the address tape as a list of choices to make during this computation.
  - If a choice is invalid, abort simulation.
  - If an accepting configuration is reached at the end of the simulation, *accept*.
(4) Increment the content of the address tape, considered as a number in base *b*, by 1. Go to step 2.

Nondeterministic Turing Machines

D works as follows:

(1) Start: input tape contains input *w*, simulation and address tape empty
(2) Copy *w* to the simulation tape and initialize the address tape with 0.
(3) Simulate one finite computation of *N* on *w* on the simulation tape.
    - Interpret the address tape as a list of choices to make during this computation.
    - If a choice is invalid, abort simulation.
    - If an accepting configuration is reached at the end of the simulation, *accept*.
(4) Increment the content of the address tape, considered as a number in base *b*, by 1. Go to step 2.

□

Enumerators

Definition 2.10

A multi-tape Turing machine M is an **enumerator** if

Enumerators

Definition 2.10

A multi-tape Turing machine M is an **enumerator** if

- M has a designated write-only **output-tape** on which a symbol, once written, can never be changed and where the head can never move left;

Enumerators

Definition 2.10

A multi-tape Turing machine M is an **enumerator** if

- M has a designated write-only **output-tape** on which a symbol, once written, can never be changed and where the head can never move left;
- M has a **marker symbol** # separating words on the output tape.

Enumerators

Definition 2.10

A multi-tape Turing machine M is an **enumerator** if

- M has a designated write-only **output-tape** on which a symbol, once written, can never be changed and where the head can never move left;
- M has a **marker symbol** # separating words on the output tape.

We define the **language generated by M** to be the set $\mathcal{G}(M)$ of all words that eventually appear between two consecutive # on the output tape of M when started on the empty word as input.

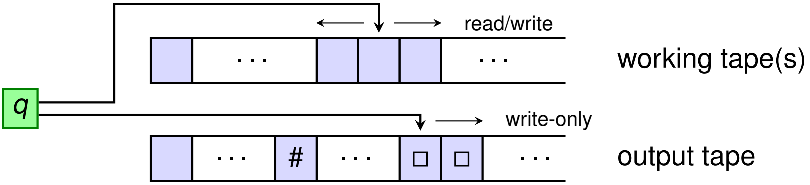
Enumerators

Definition 2.10

A multi-tape Turing machine M is an **enumerator** if

- M has a designated write-only **output-tape** on which a symbol, once written, can never be changed and where the head can never move left;
- M has a **marker symbol** # separating words on the output tape.

We define the **language generated by M** to be the set $\mathcal{G}(M)$ of all words that eventually appear between two consecutive # on the output tape of M when started on the empty word as input.



Enumerators

Theorem 2.11

A language $\mathcal{L}$ is Turing-recognisable if and only if there exists some enumerator E such that $\mathcal{G}(E) = \mathcal{L}$.

Enumerators

Theorem 2.11

A language $\mathcal{L}$ is Turing-recognisable if and only if there exists some enumerator E such that $\mathcal{G}(E) = \mathcal{L}$.

Proof.

Let E be an enumerator for $\mathcal{L}$. Then the following TM accepts $\mathcal{L}$:

$\mathcal{M} :=$ On input w

- Simulate E on the empty input. Compare every string output by E with w
- If w appears in the output of E, *accept*

Enumerators

Let $\mathcal{L} = \mathcal{L}(\mathcal{M})$ for some TM M, and let $s_1, s_2, \ldots$ be an enumeration of Σ^*.

Enumerators

Let $\mathcal{L} = \mathcal{L}(\mathcal{M})$ for some TM M, and let $s_1, s_2, \ldots$ be an enumeration of Σ^*. Then the following enumerator $\mathcal{E}$ enumerates $\mathcal{L}$:

$\mathcal{E}$:= Ignore the input.

- Repeat for $i = 1, 2, 3, \ldots$
  - Run M for i steps on each input $s_1, s_2, \ldots, s_i$
  - If any computation accepts, print the corresponding s_j followed by #

□

Enumerators

Let $\mathcal{L} = \mathcal{L}(\mathcal{M})$ for some TM M, and let $s_1, s_2, \ldots$ be an enumeration of Σ^*. Then the following enumerator $\mathcal{E}$ enumerates $\mathcal{L}$:

$\mathcal{E} :=$ Ignore the input.

- Repeat for $i = 1, 2, 3, \ldots$
  - Run M for i steps on each input $s_1, s_2, \ldots, s_i$
  - If any computation accepts, print the corresponding s_j followed by #

□

Theorem 2.12

If $\mathcal{L}$ is Turing-recognisable, then there exists an enumerator for $\mathcal{L}$ that prints each word of $\mathcal{L}$ exactly once.

Enumerators

Theorem 2.13

A language $\mathcal{L}$ is decidable if and only if there exists an enumerator for $\mathcal{L}$ that outputs exactly the words of $\mathcal{L}$ in some order of non-decreasing length.

Enumerators

Theorem 2.13

A language $\mathcal{L}$ is decidable if and only if there exists an enumerator for $\mathcal{L}$ that outputs exactly the words of $\mathcal{L}$ in some order of non-decreasing length.

Proof.

Suppose $\mathcal{L}$ to be decidable, and let M be a TM that decides $\mathcal{L}$.

Enumerators

Theorem 2.13

A language $\mathcal{L}$ is decidable if and only if there exists an enumerator for $\mathcal{L}$ that outputs exactly the words of $\mathcal{L}$ in some order of non-decreasing length.

Proof.

Suppose $\mathcal{L}$ to be decidable, and let M be a TM that decides $\mathcal{L}$.

- Define a TM M' that generates, on some scratch tape, all words over Σ in some order of non-decreasing length. (Exercise!)

Enumerators

Theorem 2.13

A language $\mathcal{L}$ is decidable if and only if there exists an enumerator for $\mathcal{L}$ that outputs exactly the words of $\mathcal{L}$ in some order of non-decreasing length.

Proof.

Suppose $\mathcal{L}$ to be decidable, and let M be a TM that decides $\mathcal{L}$.

- Define a TM M' that generates, on some scratch tape, all words over Σ in some order of non-decreasing length. (Exercise!)
- For each word w thus generated, simulate M on w_i. If M accepts w, then M' prints w followed by #.

Enumerators

Theorem 2.13

A language $\mathcal{L}$ is decidable if and only if there exists an enumerator for $\mathcal{L}$ that outputs exactly the words of $\mathcal{L}$ in some order of non-decreasing length.

Proof.

Suppose $\mathcal{L}$ to be decidable, and let M be a TM that decides $\mathcal{L}$.

- Define a TM M' that generates, on some scratch tape, all words over Σ in some order of non-decreasing length. (Exercise!)
- For each word w thus generated, simulate M on w_i. If M accepts w, then M' prints w followed by #.

Then M' enumerates exactly the words of $\mathcal{L}$ in some order of non-decreasing length.

Enumerators

Now suppose $\mathcal{L}$ can be enumerated by some TM $\mathcal{E}$ in some order of non-decreasing length.

Enumerators

Now suppose $\mathcal{L}$ can be enumerated by some TM $\mathcal{E}$ in some order of non-decreasing length.

- If $\mathcal{L}$ is finite, then $\mathcal{L}$ is accepted by a finite automaton.

Enumerators

Now suppose $\mathcal{L}$ can be enumerated by some TM $\mathcal{E}$ in some order of non-decreasing length.

- If $\mathcal{L}$ is finite, then $\mathcal{L}$ is accepted by a finite automaton.
- If $\mathcal{L}$ is infinite, then we define a decider $\mathcal{M}$ for it as follows.

Enumerators

Now suppose $\mathcal{L}$ can be enumerated by some TM $\mathcal{E}$ in some order of non-decreasing length.

- If $\mathcal{L}$ is finite, then $\mathcal{L}$ is accepted by a finite automaton.
- If $\mathcal{L}$ is infinite, then we define a decider $\mathcal{M}$ for it as follows.

$\mathcal{M}$:= On input *w*

- Simulate $\mathcal{E}$ until it either outputs *w* or some word longer than *w*
- If $\mathcal{E}$ outputs *w*, then *accept*, else *reject*.

Enumerators

Now suppose $\mathcal{L}$ can be enumerated by some TM $\mathcal{E}$ in some order of non-decreasing length.

- If $\mathcal{L}$ is finite, then $\mathcal{L}$ is accepted by a finite automaton.
- If $\mathcal{L}$ is infinite, then we define a decider $\mathcal{M}$ for it as follows.

$\mathcal{M}$:= On input w

- Simulate $\mathcal{E}$ until it either outputs w or some word longer than w
- If $\mathcal{E}$ outputs w, then *accept*, else *reject*.

Observation: since $\mathcal{L}$ is infinite, for each $w \in \Sigma^*$ the TM $\mathcal{E}$ will eventually generate w or some word longer than w.

Enumerators

Now suppose $\mathcal{L}$ can be enumerated by some TM $\mathcal{E}$ in some order of non-decreasing length.

- If $\mathcal{L}$ is finite, then $\mathcal{L}$ is accepted by a finite automaton.
- If $\mathcal{L}$ is infinite, then we define a decider $\mathcal{M}$ for it as follows.

$\mathcal{M}$:= On input w

- Simulate $\mathcal{E}$ until it either outputs w or some word longer than w
- If $\mathcal{E}$ outputs w, then *accept*, else *reject*.

Observation: since $\mathcal{L}$ is infinite, for each $w \in \Sigma^*$ the TM $\mathcal{E}$ will eventually generate w or some word longer than w. Therefore, $\mathcal{M}$ always halts and thus decides $\mathcal{L}$.

Enumerators

Now suppose $\mathcal{L}$ can be enumerated by some TM $\mathcal{E}$ in some order of non-decreasing length.

- If $\mathcal{L}$ is finite, then $\mathcal{L}$ is accepted by a finite automaton.
- If $\mathcal{L}$ is infinite, then we define a decider $\mathcal{M}$ for it as follows.

$\mathcal{M}$:= On input w

- Simulate $\mathcal{E}$ until it either outputs w or some word longer than w
- If $\mathcal{E}$ outputs w, then *accept*, else *reject*.

Observation: since $\mathcal{L}$ is infinite, for each $w \in \Sigma^*$ the TM $\mathcal{E}$ will eventually generate w or some word longer than w. Therefore, $\mathcal{M}$ always halts and thus decides $\mathcal{L}$.

□

Summary and Outlook

Turing Machines are a simple model of computation

Recognisable (semi-decidable) = recursively enumerable

Decidable = computable = recursive

Many variants of TMs exist – they normally recognise/decide the same languages

What's next?

- A short look into undecidability
- Recursion and self-referentiality
- Actual complexity classes