Advanced Topics in Theoretical Computer Science

Part 1: Turing Machines and Turing Computability

25.10.2012

Viorica Sofronie-Stokkermans

Universität Koblenz-Landau

e-mail: sofronie@uni-koblenz.de

Turing Machines

Overview: Turing Machines

- Accept languages of type 0.
- First memory: state (finite)
- Second memory: tape
  unlimited size; access at arbitrary place.
- Have a read/write head which can move left/right over the tape.
- Input word: initially on the tape.
  The machine can read it arbitrarily often.

Turing Machines

Definition (Deterministic Turing Machine (DTM))

A deterministic Turing Machine (DTM) $\mathcal{M}$ is a tuple

$$\mathcal{M} = (K, \Sigma, \delta, s)$$

where:

- K is a finite set of states with $h \notin K$;
  (h is the halting state)
- Σ is an alphabet with $L, R \notin \Sigma, \# \in \Sigma$
- $\delta : K \times \Sigma \to (K \cup \{h\}) \times (\Sigma \cup \{L, R\})$ is a transition function
- $s \in K$ is an initial state

Number of states: $|K| - 1$ (initial state is not counted)

Turing Machines

Attention: Various definitions for Turing machines in the literature.

Some definitions do not require δ to be a total function

Same expressive power (but a different definition for “hanging”)

Turing Machines

Attention: Various definitions for Turing machines in the literature.

Some definitions do not require δ to be a total function

Same expressive power (but a different definition for “hanging”)

Here: We require δ to be totally defined.

Turing Machine

Example:

state	$\#$	1	c
q_0	(q_1, c)	$-$	$-$
q_1	(q_2, R)	(q_1, L)	(q_1, L)
q_2	$-$	$(q_3, \#)$	$(q_7, \#)$
q_3	(q_4, c)	$-$	$-$
q_4	$(q_5, 1)$	(q_4, R)	(q_4, R)
q_5	$(q_6, 1)$	(q_5, L)	(q_5, L)
q_6	$-$	(q_1, L)	$-$
q_7	(q_8, R)	$-$	$-$
q_8	$(h, \#)$	(q_8, R)	$-$

Positions marked with $-$:

values which are never used during the execution.

Definition of TM in which δ is partially defined:

$-$ means "undefined"

Definition of TM im which δ is totally defined:

$-$ can e.g. mean $\delta(x) = x$ for that input (loop)

Turing Machines

How does a Turing Machine work?

Transition $\delta(q, a) = (q', x)$ means:

Depending on the:

- current state $q \in K$
- symbol $a \in \Sigma$ on which the read/write head is positioned

the following happens:

- a step to the left (if $x = L$)
- a step to the right (if $x = R$)
- the symbol a which currently stands below the read/write head is overwritten with symbol $b \in \Sigma$ (if $x = b \in \Sigma$)
- the state is changed to $q' \in K \cup \{h\}$.

Turing Machine

The tape

The tape of a DTM is unlimited on one side:

- infinitely long to the right
- has an end on the left
- when a DTM tries to go beyond the left end, it remains “hanging”. In this case the computation does not halt.

Turing Machines

Configuration

- A configuration describes the complete current situation of a machine in a computation.
- A computation is a sequence of configurations,
  where there is always a transition from a configuration to the next configuration.

The configuration $s, \#w\underline{a}u\#$ of a DTM consists of 4 elements:

- current state s
- word w at the left of the read/write head
- the symbol a on which the head is placed
- the word u at the right of the actual head position

Remark: The tape has only finitely many symbols which are not blanks

Turing Machine

Initial configuration

- to the left of the tape: blank
- directly right of this blank: input word
- If a DTM receives several words $w_1, \ldots, w_n$ after each other, they are separated by blanks:

$$\#w_1\#w_2\ldots\#w_n\#$$

- To the right of the last input word there are only blanks
- the read/write head of the DTM is positioned on the blank directly to the right after the last input word

$$\#w_1\#w_2\ldots\#w_n\underline{\#}$$

- The machine is in the initial state s

Turing Machines

Empty symbol: The special symbol $\#$ (blank) is the empty symbol.

This symbol is never part of the input word; it can for instance be used to separate words on the tape.

Turing Machines

Definition (Input)

A word w is called an input for $\mathcal{M}$, if $\mathcal{M}$ starts with the start configuration

$$C_0 = s, \#w\underline{\#}$$

$(w_1, \dots, w_n)$ is an input for $\mathcal{M}$, if $\mathcal{M}$ starts with the start configuration

$$C_0 = s, \#w\#w_2\# \dots \#w_n\underline{\#}$$

Turing Machines

Definition (Transition from a configuration to another configuration)
Let $C = q, w\underline{a}u$ be a configuration.

- If $\delta(q, a) = (q', b)$, we have a transition $C \vdash C'$ where

$$C' = q', w\underline{b}u$$

- If $\delta(q, a) = (q', L)$ and $w \neq \epsilon$, we have a transition $C \vdash C'$ where C' is like C, but the head is moved with one position to the left.
- If $\delta(q, a) = (q', R)$, we have a transition $C \vdash C'$ where C' is like C, but the head is moved with one position to the right.

Remark: If $C = q, w\underline{a}u$, with $w = \epsilon$ and $\delta(q, a) = (q', L)$
there can be no transition to another configuration.

Turing Machines

Definition (To halt, to hang)

Let $\mathcal{M}$ be a Turing machine.

- $\mathcal{M}$ halts in $C = q, w\underline{a}u$ iff $q = h$
- $\mathcal{M}$ hangs in $C = q, w\underline{a}u$ iff there is no next configuratiion (especially when $w = \epsilon$ and $\exists q' \;\; \delta(q, a) = (q', L)$).

Remark: For the definition of TM in which δ is partially defined, $\mathcal{M}$ hangs in $C = q, w\underline{a}u$ also if $\delta(q, a)$ is undefined.

Turing Machine

Definition (Computation)

Let $\mathcal{M}$ be a Turing machine. We write

$$C \vdash_{\mathcal{M}}^{*} C'$$

iff there exists a sequence of configurations

$$C_0, C_1, \ldots, C_n \qquad (n \geq 0)$$

such that:

- $C = C_0$ and $C' = C_n$
- for all $i < n$, $C_i \vdash_{\mathcal{M}} C_{i+1}$

Then $C_0 C_1 \ldots C_n$ is a computation of length n from C_0 to C_n.

Constructing Turing Machines

Assume we can construct “simple” Turing machines, which perform simple computations

Goal: Design TM for a complex computation task

A possible approach:

- Describe steps which would lead to the desired computation
- Turing Machines for individual steps
- Consecutive steps are combined using “compositions” of Turing machines

$\mathcal{M}_1 \rightarrow \mathcal{M}_2$ is a Turing machine which first works as $\mathcal{M}_1$ and then, if $\mathcal{M}_1$ halts, continues working as $\mathcal{M}_2$.

Turing Machine

Example:

Steps necessary for constructing a DTM $\mathcal{C}$ which receives as an input a string over $\{1\}$ and copies it, i.e.

> If at the beginning there are n ones on the tape, after the execution of $\mathcal{C}$ there are $2n$ ones on the tape (separated by a blank $\#$).

(1) Go to the beginning of the word.

(2) Go right.

(3) Read symbol. If symbol is 1 replace it with $\#$ and go right until reading a $\#$ (initial word ends); then right until reading a $\#$ (end of the copied sequence)

(4) Write 1 on tape, then move left until reading a $\#$ (space between word and copy); then left until reading a $\#$ (symbol which was replaced with $\#$ in (3))

(5) Write back 1 at that place and goto step (2).

(6) If in (3) $\#$ is read copying is finished; go right until reading a $\#$.

Turing Machines can compute functions

Definition (TM-computable function)

Let Σ_0 be an alphabet with $\# \not\in \Sigma_0$.

A partial function

$$F : (\Sigma_0^*)^m \to (\Sigma_0^*)^n$$

is DTM-computable if there exists a deterministic Turing machine

$$\mathcal{M} = (K, \Sigma, \delta, s)$$

- with $\Sigma_0 \subseteq \Sigma$
- such that for all $w_1, \ldots, w_m, u_1, \ldots, u_n \in \Sigma_0^*$ the following hold:

(1) $f(w_1, \ldots, w_m) = (u_1, \ldots, u_n)$ iff

$$s, \#w_1\# \ldots \#w_m\underline{\#} \vdash_{\mathcal{M}}^* h, \#u_1\# \ldots \#u_n$$

(2) $f(w_1, \ldots, w_m)$ is undefined iff

$\mathcal{M}$ started with $s, \#w_1\# \ldots \#w_m\underline{\#}$ does not halt (i.e. it runs forever or it hangs).

Turing Machines can compute functions

Attention

We consider Turing Machines in a different way from the way automata are considered:

- For finite automata and push-down automata: one studies which languages they accept.
- For Turing Machines we study
  - which languages they accept and
  - which functions they compute

Acceptance of a language is a special case of function computation.

Turing Machine: Accepted language

Definition

- A word w is accepted by a DTM $\mathcal{M}$ if $\mathcal{M}$ halts on input w (such that at the end, the head is positioned on the first blank on the right of w)

- A language $L \subseteq \Sigma^*$ is accepted by a DTM $\mathcal{M}$ iff the words from L (and no other words) are accepted by $\mathcal{M}$.

Attention:

For words which are not accepted, the DTM does not need to halt

(it is not allowed to halt, in fact)

Turing Machines: Functions on natural numbers

Functions on natural numbers

- We use the unary representation: a number is represented on the tape as a string of n vertical lines.
- A Turing Machine computes a function

$$f : \mathbb{N}^k \to \mathbb{N}^n$$

in unary representation as follows:

(1) if $f(i_1, \ldots, i_k) = (j_1, \ldots, j_n)$, then $\mathcal{M}$ computes

$$s, \#|^{i_1}\#|^{i_2}\# \ldots \#|^{i_k}\underline{\#} \vdash^*_{\mathcal{M}} h, \#|^{j_1}\#|^{j_2}\# \ldots \#|^{j_n}\underline{\#}.$$

(2) if $f(i_1, \ldots, i_k)$ is undefined then $\mathcal{M}$ halts not on input $s, \#|^{i_1}\#|^{i_2}\# \ldots \#|^{i_k}\underline{\#}$.

Turing Machines: Functions on natural numbers

Definition

- TM^{part} is the set of all partial TM-computable functions $f : \mathbb{N}^k \longrightarrow \mathbb{N}$
- TM is the set of all total TM-computable functions $f : \mathbb{N}^k \longrightarrow \mathbb{N}$

Turing Machines: Functions on natural numbers

Definition

- TM^{part} is the set of all partial TM-computable functions $f : \mathbb{N}^k \longrightarrow \mathbb{N}$
- TM is the set of all total TM-computable functions $f : \mathbb{N}^k \longrightarrow \mathbb{N}$

Remark: Restrictions when defining TM and TM^{part}:

- Only functions over $\mathbb{N}$
- Only functions with values in $\mathbb{N}$ (not in $\mathbb{N}^m$)

Turing Machines: Functions on natural numbers

Definition

- TM^{part} is the set of all partial TM-computable functions $f : \mathbb{N}^k \longrightarrow \mathbb{N}$
- TM is the set of all total TM-computable functions $f : \mathbb{N}^k \longrightarrow \mathbb{N}$

Remark: Restrictions when defining TM and TM^{part}:

- Only functions over $\mathbb{N}$
- Only functions with values in $\mathbb{N}$ (not in $\mathbb{N}^m$)

This is not a real restriction:

Words from other domains can be encoded as natural numbers.

Other types of Turing machines

Standard deterministic Turing Machines (Standard DTM)

The Turing machines defined before

- are deterministic
- have a tape which is infinite on one side.

We will call such machines Standard Turing Machines
(Standard DTM, or DTM for short)

Other types of Turing machines

Standard deterministic Turing Machines (Standard DTM)

The Turing machines defined before

- are deterministic
- have a tape which is infinite on one side.

We will call such machines Standard Turing Machines
(Standard DTM, or DTM for short)

Other types of Turing machines:

- Tape infinite on both sides
- Several tapes
- Non-deterministic Turing machines

Turing machines with both sides infinite tape

- The definition of a machine remains the same
- The definition of a configuration changes:
  Configurations: $q, w\underline{a}u$, but:
  - w consists of all symbols until the last non-blank symbol on the left of reading head
  - u consists of all symbols until the last non-blank symbol on the right of reading head

  $w = \epsilon$: only blanks on the left of the read/write head
  $u = \epsilon$: only blanks on the right of the read/write head
- Computations defined as for TM's (taking into account the different definition for configuration)

Turing machines with both sides infinite tape

Theorem

For every TM with both sides infinite tape which computes a function f or accepts a language L, there exists a standard DTM $\mathcal{M}'$ which also computes f (resp. accepts L).

Turing machines with several tapes

Definition (DTM with k (half-)tapes; k-Turing machine; k-DTM)
A Turing Machine $\mathcal{M} = (K, \Sigma_1, \dots, \Sigma_k, \delta, s)$ with k (half-)tapes (each with a read/write head) is a Turing Machine with a transition function:

$$\delta : K \times \Sigma_1 \times \cdots \times \Sigma_k \to (K \cup \{h\}) \times (\Sigma_1 \cup \{R, L\}) \times \cdots \times (\Sigma_k \cup \{R, L\})$$

A configuration of a k-Turing machine has the form:

$$C = q,\ w_1\underline{a_1}u_1,\ w_2\underline{a_2}u_2, \dots,\ w_k\underline{a_k}u_k$$

- The heads can move independently (otherwise we would only have a DTM with k tracks)
- Definition of computation: analogous to that for Standard-DTM
- For a k-DTM which computes a function $f : \Sigma_0^m \to \Sigma_0^n$ we make the convention that input/output takes place on the first tape.

Turing machines with several tapes

Theorem
For every k-DTM which computes a function f (or accepts a language L) there exists a DTM $\mathcal{M}'$ which computes f (resp. accepts L).

Non-deterministic Turing machines

Definition

A non-deterministic Turing machine $\mathcal{M}$ is a tuple:

$$\mathcal{M} = (K, \Sigma, \Delta, s)$$

where:

- K, Σ and s are as for deterministic Turing machines.
- Δ is a transition relation:

$$\Delta \subseteq (K \times \Sigma) \times ((K \cup \{h\}) \times (\Sigma \cup \{L, R\}))$$

Non-deterministic Turing machines

Configurations: defined as for DTMs.

For non-deterministic Turing machines it is possible that there are several ways of evolving from a given configuration.

Non-deterministic Turing machines

Definitions

Let $\mathcal{M}$ be a non-deterministic Turing machine.

- $\mathcal{M}$ halts on input w if among all possible computations which $\mathcal{M}$ can choose, there exists at least one for which $\mathcal{M}$ reaches a halting configuration.
- $\mathcal{M}$ hangs in a configuration C if there is no configuration into which $\mathcal{M}$ can move from C according to Δ.
- $\mathcal{M}$ accepts a word w iff $\mathcal{M}$ can reach from $s, \#w\underline{\#}$ a halting state.
- $\mathcal{M}$ accepts a language L iff the words from L (and no other words) are accepted by $\mathcal{M}$.

Non-deterministic Turing machines

DTM were also used to compute functions

Can NTMs be also used for computing functions?

Non-deterministic Turing machines

DTM were also used to compute functions

Can NTMs be also used for computing functions?

Problem. For computing a function it is not only important that the Turing machine halts, but also with which contents on the tape:

Which of the many halting configurations should hold?

Non-deterministic Turing machines

DTM were also used to compute functions

Can NTMs be also used for computing functions?

Problem. For computing a function it is not only important that the Turing machine halts, but also with which contents on the tape:

Which of the many halting configurations should hold?

To avoid this problem, we do not extend the notions of “decide” and “enumerate” to NTM.

In general, NTMs will not be used to compute functions.

Non-deterministic Turing machines

How does a non-deterministic Turing machine compute?

- The rules of a DTM can be seen as a “program” (consisting of very simple steps).
- This is different for NTMs
- A NTM is not a machine which guesses always right.

Non-deterministic Turing machines

How does a non-deterministic Turing machine compute?

- The rules of a DTM can be seen as a “program” (consisting of very simple steps).
- This is different for NTMs
- A NTM is not a machine which guesses always right.

Correct intuition

- Transitions from configurations to successor configurations correspond to the set Δ of transition rules
- In addition: Search

Alternative formulation: A NTM considers all alternatives in parallel.

An NTM accepts a word if there is at least one computation which ends in a halting configuration. [This is sometimes expressed as **“the NTM guesses”** – (but care is needed with this terminology!)]

Non-deterministic Turing Machines

Example:

Let $L = \{|^n \mid n \text{ not prime and } n \geq 2\}$

An NTM can accept this language as follows:

- "Guess" a number a write (left) on the tape.
- "Guess" a second number and write it.
- Multiply the two numbers.
- Compare the result with the input.
- Stop iff the result is equal to the input (in halting state).

Non-deterministic Turing Machines

Example:

Let $L = \{|^n \mid n \text{ not prime and } n \geq 2\}$

An NTM can accept this language as follows:

- "Guess" a number and write it on the tape.
  (consider all possible numbers smaller than n in parallel)
- "Guess" a second number and write it.
  (consider all possible numbers smaller than n in parallel)
- Multiply the two numbers.
- Compare the result with the input.
- Stop iff the result is equal to the input (in halting state).

Non-deterministic Turing machines

Theorem

Every language which is accepted by some non-deterministic Turing machine is also accepted by a standard deterministic Turing machine.

Universal Turing machines

Comparison between Turing machines and “normal” computer

Turing machines are very powerful.

Universal Turing machines

Comparison between Turing machines and "normal" computer

Turing machines are very powerful.

How powerful?

- A Turing machine has a given "program" (set of rules, δ)
- "Normal" computer can execute arbitrary programs.

Universal Turing machines

Comparison between Turing machines and "normal" computer

Turing machines are very powerful.

How powerful?

- A Turing machine has a given "program" (set of rules, δ)
- "Normal" computer can execute arbitrary programs.

Actually, this is possible also with Turing machines

Universal Turing Machines

Turing machine which simulates other Turing machines

- Universal Turing machine $\mathcal{U}$ receives as input
  (i) the rules of an arbitrary TM $\mathcal{M}$ and
  (ii) a word w.
- $\mathcal{U}$ simulates $\mathcal{M}$, by always changing the configurations (according to the transition function δ) the way $\mathcal{M}$ would change them.

Universal Turing Machines

Turing machine which simulates other Turing machines

- Universal Turing machine $\mathcal{U}$ receives as input
  (i) the rules of an arbitrary TM $\mathcal{M}$ and
  (ii) a word w.
- $\mathcal{U}$ simulates $\mathcal{M}$, by always changing the configurations (according to the transition function δ) the way $\mathcal{M}$ would change them.

Problem: Turing machines take words (or numbers) as inputs.

Question:
Can we encode an arbitraty Turing machine as a number or as a word?

Universal Turing Machines

Gödelisation

Method for assigning with every Turing machine a number or a word (Gödel number or Gödel word) such that the Turing machine can be effectively reconstructed from that number (or word).

Universal Turing Machines

Gödelisation

Method for assigning with every Turing machine a number or a word (Gödel number or Gödel word) such that the Turing machine can be effectively reconstructed from that number (or word).

We can construct a universal Turing machine.

Acceptance and Decidability

We now formalize notions such as:

- Acceptable language
- Recursively enumerable language
- Enumerable language
- Decidable language

and present the relationships between these notions.

Acceptance and Decidability

Acceptance

A DTM $\mathcal{M}$ accepts a language L if

- for every input word $w \in L$, $\mathcal{M}$ halts;
- for every input word $w \notin L$, $\mathcal{M}$ computes infinitely or hangs.

Acceptance and Decidability

Acceptance

A DTM $\mathcal{M}$ accepts a language L if

- for every input word $w \in L$, $\mathcal{M}$ halts;
- for every input word $w \notin L$, $\mathcal{M}$ computes infinitely or hangs.

Deciding

A DTM $\mathcal{M}$ decides a language L if

- for every input word $w \in L$, $\mathcal{M}$ halts with band contents Y (yes)
- for every input word $w \notin L$, $\mathcal{M}$ halts with band contents N (no)

Acceptance and Decidability

Definition (Decidable language)

Let L be a language over Σ_0 with $\#, Y, N \notin \Sigma_0$.

Let $\mathcal{M} = (K, \Sigma, \delta, s)$ be a DTM with $\Sigma_0 \subseteq \Sigma$.

- $\mathcal{M}$ decides L if for all $w \in \Sigma_0^*$:

$$s, \#w\underline{\#} \vdash_{\mathcal{M}}^* \begin{cases} h, \#Y\underline{\#} & \text{if } w \in L \\ h, \#N\underline{\#} & \text{if } w \notin L \end{cases}$$

- L is called decidable if there exists a DTM which decides L.

Acceptance and Decidability

Definition (Acceptable language)
Let L be a language over Σ_0 with $\#, Y, N \notin \Sigma_0$.
Let $\mathcal{M} = (K, \Sigma, \delta, s)$ be a DTM with $\Sigma_0 \subseteq \Sigma$.

- $\mathcal{M}$ accepts a word $w \in \Sigma_0^*$ if $\mathcal{M}$ always halts on input w.
- $\mathcal{M}$ accepts the language L if for all $w \in \Sigma_0^*$, $\mathcal{M}$ accepts w iff $w \in L$.
- L is called acceptable (or semi-decidable) if there exists a DTM which accepts L.

Recursively enumerable

Definition (Recursively enumerable language)

Let L be a language over Σ_0 with $\#, Y, N \notin \Sigma_0$.
Let $\mathcal{M} = (K, \Sigma, \delta, s)$ be a DTM with $\Sigma_0 \subseteq \Sigma$.

- $\mathcal{M}$ enumerates L if there exists a state $q_B \in K$ (the blink state) such that:

$$L = \{w \in \Sigma_0^* \mid \exists u \in \Sigma^*; s, \underline{\#} \vdash_{\mathcal{M}}^* q_B, \#w\underline{\#}u\}$$

- L is called recursively enumerable if there exists a DTM $\mathcal{M}$ which enumerates L.

Recursively enumerable

Attention: recursively enumerable $\neq$ enumerable!

Recursively enumerable

Attention: recursively enumerable $\neq$ enumerable!

Difference:

- ***L* enumerable:** there exists a surjective map of the natural numbers onto L.
- ***L* recursively enumerable:** the surjective map can be computed by a Turing machine.

Because of the finiteness of the words and of the alphabet, all languages are enumerable. But not all languages are recursively enumerable.

Recursively enumerable

Attention: recursively enumerable $\neq$ enumerable!

Difference:

- L **enumerable:** there exists a surjective map of the natural numbers onto L.
- L **recursively enumerable:** the surjective map can be computed by a Turing machine.

Because of the finiteness of the words and of the alphabet, all languages are enumerable. But not all languages are recursively enumerable.

$\mapsto$ Set of all languages is not enumerable;
Turing machines can be enumerated.

Recursively enumerable

Attention: recursively enumerable $\neq$ decidable!

Recursively enumerable

Attention: recursively enumerable $\neq$ decidable!

Examples:

The following sets are recursively enumerable, but not decidable:

- The set of the Gödelisations of all halting Turing machines.
- The set of all terminating programs.
- The set of all valid formulae in predicate logic.

Undecidability of the halting problem

$\mathcal{M}$ Turing machine $\mapsto G(\mathcal{M})$ Gödelisation

$$HALT = \{G(\mathcal{M})w \mid \mathcal{M} \text{ halts on input } w\}$$

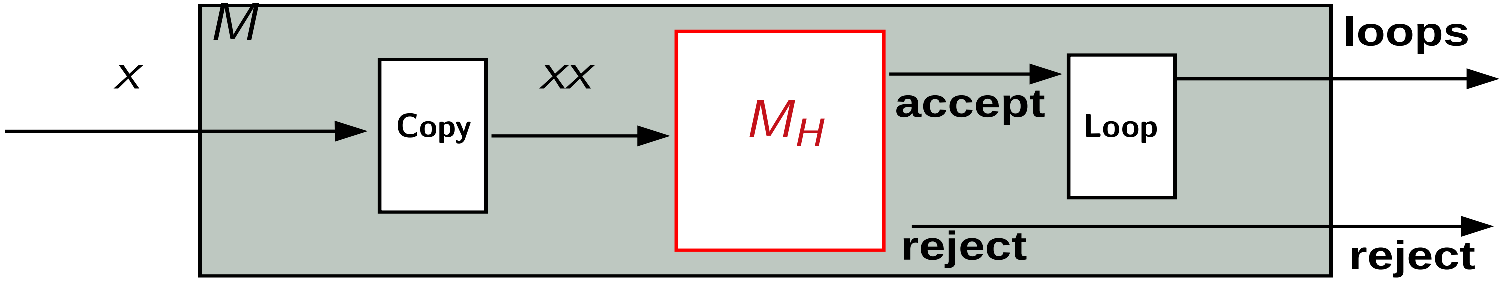
Is *HALT* decidable?

Undecidability of the halting problem

Proposition: $HALT = \{G(\mathcal{M})w \mid \mathcal{M}$ halts on input $w\}$ is not decidable.

Proof: Assume, in order to derive a contradiction, that there exists a TM M_H which halts on every input and accepts only inputs in $HALT$.

We construct the following TM:



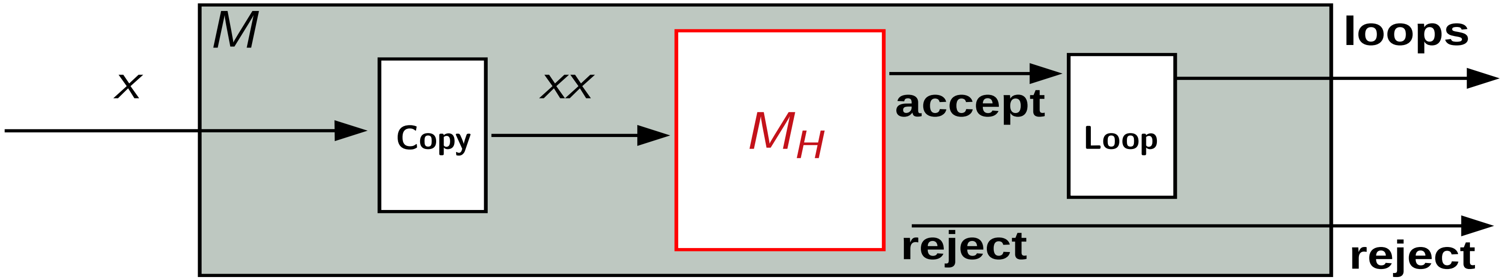
1. Let x be the input.
2. Copy the input. Let xx be the result.
3. Decide using M_H if $xx \in HALT$
4.   If yes: write infinitely many 1s to the right.
5.   If no: halt

Undecidability of the halting problem

Proposition: $HALT = \{G(\mathcal{M})w \mid \mathcal{M}$ halts on input $w\}$ is not decidable.

Proof: Assume, in order to derive a contradiction, that there exists a TM M_H which halts on every input and accepts only inputs in $HALT$.

What happens when we start M with input $G(M)$?



Case 1: M started with $G(M)$ halts: Then $G(M)G(M) \notin HALT$ Contradiction!

Fall 2: M started with $G(M)$ does not halt: Then $G(M)G(M) \in HALT$ Contradiction!

Undecidability proofs

Proof via reduction

- L_1, L_2 languages
- L_1 known to be undecidable
- To show: L_2 undecidable
- Idea: Assume L_2 decidable. Let M_2 be a TM which decides L_2. Show that then we can construct a TM which decides L_1.

  For this, we have to find a computable function f which transforms an instance of L_1 into an instance of L_2

  $$\forall w (w \in L_1 \text{ iff } f(w) \in L_2)$$

  Let M_f be the TM which computes f. Construct $M_1 = M_f M_2$. Then M_1 decides L_1.

Acceptable/Recursively enumerable/Decidable

Theorem (Acceptable = Recursively enumerable)
A language is recursively enumerable iff it is acceptable.

Proposition
Every decidable language is acceptable.

Proposition
The complement of any decidable language is decidable.

Proposition (Characterisation of decidability)
A language L is decidable iff L and its complement are acceptable.

Recursively enumerable = Type 0

Formal languages are of type 0 if they can be generated by arbitrary grammars (no restrictions).

Proposition

The recursively enumerable languages (i.e. the languages acceptable by DTMs) are exactly the languages generated by arbitrary grammars (i.e. languages of type 0).