Advanced Topics in Computation Theory (COMP6702)

Nan Guan

The Hong Kong Polytechnic University

nan.guan@polyu.edu.hk

2017- 2018, Semester 1

Lecture 1: Computation Model

Outline

- Formal Languages
- Model of Computation: Turing Machine
- Decidability, Undecidability, Semi-Decidability

Problems and Computation

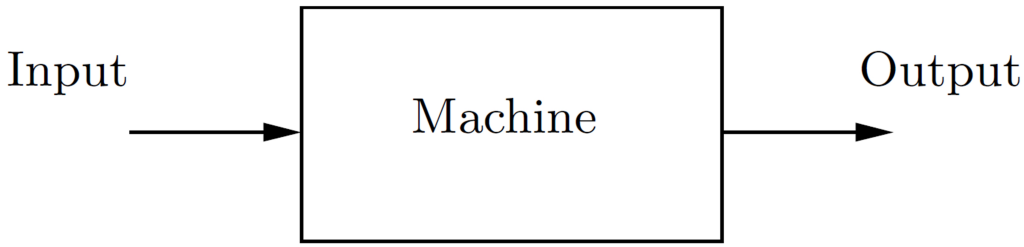


Figure: high-level model of computation

Problems as Formal Languages

- Assume a *machine* with *input* and *output*
- Formal notation for format:
  - $\Sigma = \{\sigma_1, \cdots, \sigma_k\}$ is the *alphabet*, which is a finite set of *symbols*
  - $w = w_1, \cdots, w_l$ is a *word over* Σ: $\forall i, w_i \in \Sigma$
    - ★ Write also just as $w_1 w_2 \cdots w_l$
  - $|w|$ denotes the *length* of w, the number of symbols in w
  - ε is the *empty word*, i.e., $|\varepsilon| = 0$
  - Σ^k is the set of words of length k
  - $\Sigma^* = \bigcup_{k \geq 0} \Sigma^k$ is the set of all words over Σ
  - $L_1, L_2 \subseteq \Sigma^*$, language operations:
    - ★ $L_1 \cup L_2$ (*union*)
    - ★ $L_1 \cap L_2$ (*intersection*)
    - ★ $\overline{L} := \Sigma^* - L$ (*complement*)
    - ★ $L_1 L_2 := \{w \mid \exists w_1 \in L_1, w_2 \in L_2 : w = w_1 w_2\}$ (*concatenation*)

Problems as Formal Languages (Example)

Example (NAT)

- Let $\Sigma := \{0, 1, \cdots, 9\}$
- Σ^* is the set of all strings with digits
- Let $[n]_{10}$ denote the decimal representation of $n \in \mathbb{N}$
- NAT $:= \{[n]_{10} | n \in \mathbb{N}\} \subsetneq \Sigma^*$ all representations of naturals
  - $010 \in \Sigma^* -$ NAT
- Machine for calculating square:

$$\textbf{Input}: w = [n]_{10} \in \Sigma^*$$

$$\textbf{Onput}: v \in \Sigma^* \text{ with } v = [n^2]_{10}$$

(Plus error-handling for $w \notin$ NAT)

Problems as Formal Languages (2nd Example)

Example (PRIMES)

- Let $\Sigma := \{0, 1, \cdots, 9\}$ and NAT as before
- PRIMES $:= \{[p]_{10} \mid p \text{ is a prime number}\}$
  - PRIMES $\subsetneq$ NAT
- Machine M for checking primality:

$$\textbf{Input}: w = [n]_{10} \in \Sigma^*$$

$$\textbf{Onput}: 1 \text{ if } n \text{ is prime, } 0 \text{ otherwise}$$

- This is a *decision problem*
  - PRIMES are the *positive instances*
  - Σ^* – PRIMES (everything else) the negative instances
  - M has to distinguish both (it *decides* PRIMES)

Important concept, will come back to that later!

Model of Computation

- Input/Output format defined. What else?
- Model of Computation should:
  - Define what we mean by "computation", "actions", ...
  - Be *simple* and *easy* to use
  - ... but yet powerful
- Many models:
  - Recursive Functions, Rewriting Systems, λ-Calculus, Turing Machine, ...
- All equally powerful:

Church's Thesis

All "solvable" problems can be solved by any of the above formalisms.

Turing Machine

- Turing Machines are like simplified computers containing:
  - A *tape* to read/write on
    - ★ Contain squares with one symbol each
    - ★ Used for input, output and temporary storage
    - ★ Infinitely long (in both directions)
  - A read/write *head*
    - ★ Can change the symbol on the tape at current position
    - ★ Move step by step in either direction
  - A *finite state machine*
    - ★ Including an initial state and final states
- Simple, but very powerful
- Standard model for this course

Turing Machine

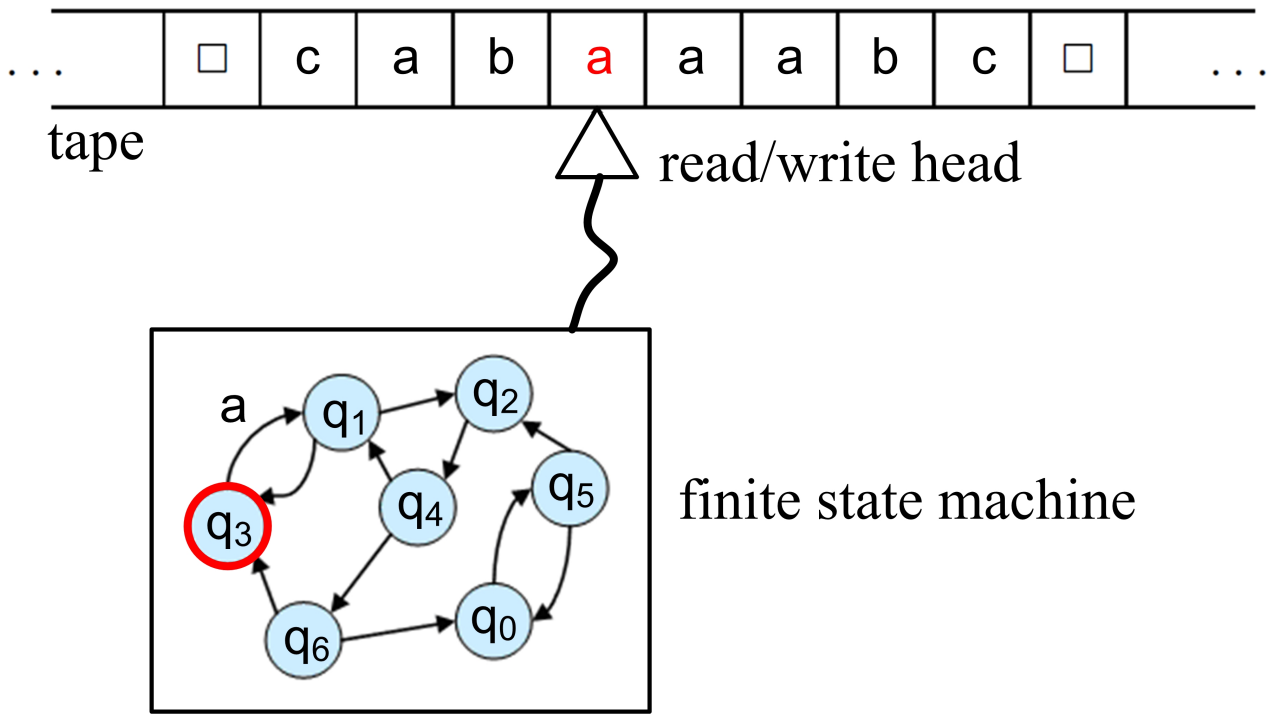


Figure: Turing Machine

Turing Machine

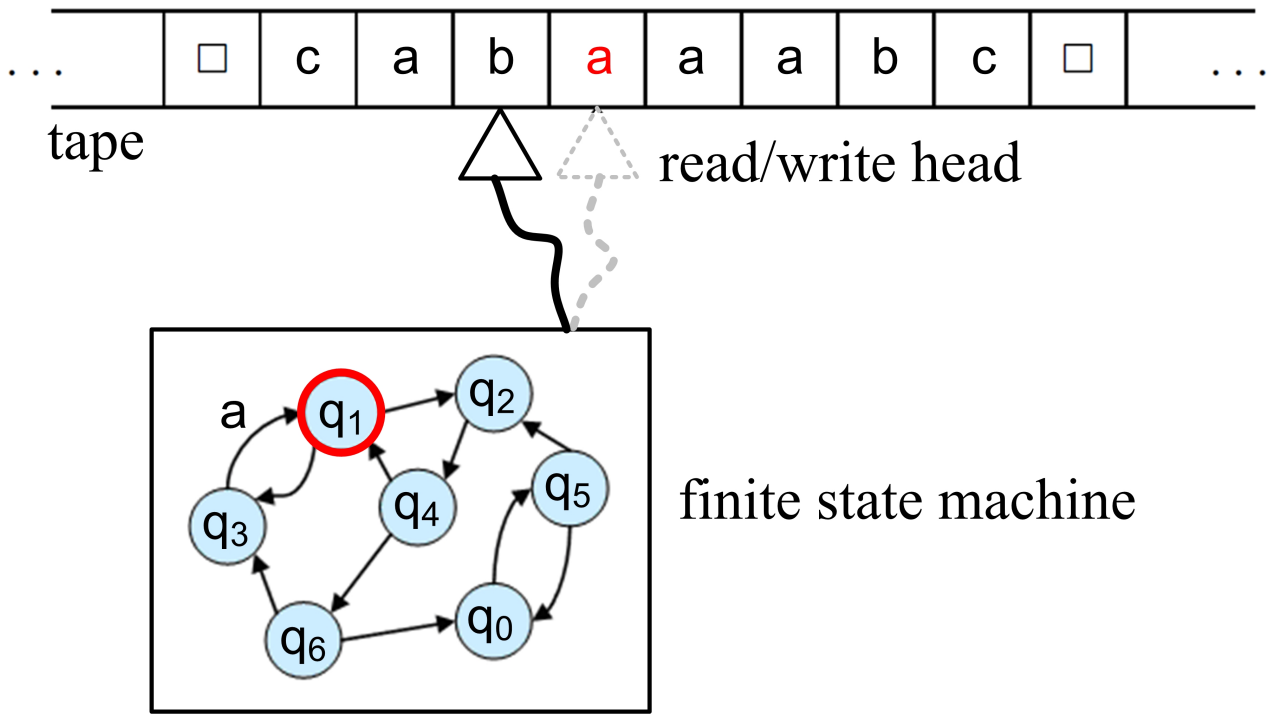


Figure: Turing Machine

Turing Machine: Definition

Definition (Turing Machine)

A Turing Machine M is a five-tuple $M = (Q, \Gamma, \delta, q_0, F)$ where

Q : a finite set of *states*

Γ : the *tape alphabet* (*alphabet* for short) including the blank: $\square \in \Gamma$

q_0 : the *initial state*, $q_0 \in Q$

F : the *set of final states*, $F \subset Q$

δ : the *transition function*, $\delta : (Q - F) \times \Gamma \to Q \times \Gamma \times \{R, N, L\}$

Operation:

- Start in state q_0, input w is on tape, head over its first symbol
- Each step:
  - Read current state q and symbol a at current position
  - Lookup $\delta(q, a) = (p, b, D)$
  - Change to state p, write b, move according to D
- Stop as soon as $q \in F$. What's left on the tape is the Output.

Turing Machine: Configuration

- *Configuration* (w, q, v) denotes the status after each step:
  - Tape contains wv (with infinitely many $\square$ around)
  - Head is over the first symbol of v
  - Machine is in state q

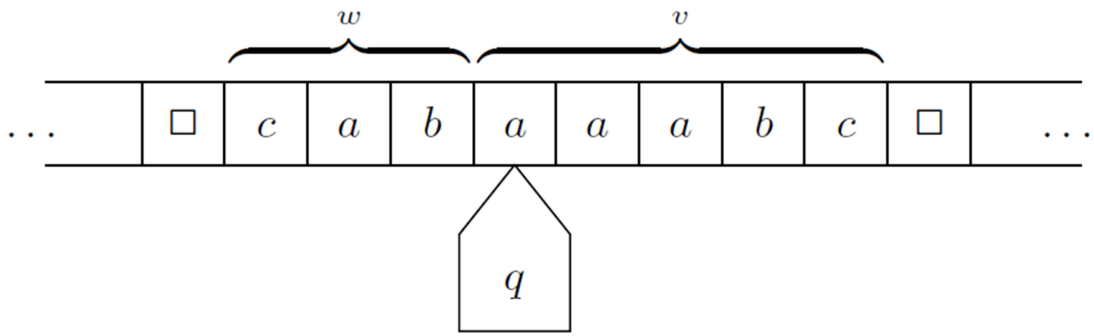


Figure: Turing machine configuration (w, q, v) with $w = cab$ and $v = aaabc$

- Start configuration: (ε, q_0, w) if input is w
- End configuration: (v, q, z) for $q \in F$
  - Output is z, denoted by $M(w)$
  - In case machine *does not halt*: $M(w) = \nearrow$

Turing Machine: Step Relation

- Step relation: Formalizes semantics of Turing Machine

Definition (Step Relation)

Let $M = (Q, \Gamma, \delta, q_0, F)$, define $\vdash$ for all $w, v \in \Gamma^*$, $a, b \in \Gamma$ and $q \in Q$ as:

$$(wa, q, bv) \vdash \begin{cases} (wac, p, v) & \text{if } \delta(q, b) = (p, c, R), \\ (wa, p, cv) & \text{if } \delta(q, b) = (p, c, N), \\ (w, p, acv) & \text{if } \delta(q, b) = (p, c, L). \end{cases}$$

- α reaches β in 1 step: $\alpha \vdash \beta$
- α reaches β in k steps: $\alpha \vdash^k \beta$
- α reaches β in any number of steps: $\alpha \vdash^* \beta$

Turing Machine: Transducers and Acceptors

- Definition so far: Receive input, compute output
- We call this a *transducer*:
  - Interpret a Turing Machine M as a function $f : \Sigma^* \to \Sigma^*$
  - All such f are called *computable functions*
  - *Partial functions* may be undefined for some inputs w
  - *Total functions* are defined for all inputs
- For decision problems L: Only want a positive or negative answer
- We call this an *acceptor*:
  - M halts in
    - ★ either state q_{yes} for positive instances $w \in L$
    - ★ or state q_{no} for negative instances $w \notin L$
  - Output does not matter, only final state
  - M *accepts* the language $L(M)$:

$$L(M) := \{w \in \Sigma^* | \exists y, z \in \Gamma^* : (\varepsilon, q_0, w) \vdash^* (y, q_{yes}, z)\}$$

- We mainly focus on decision problems and acceptors in this course.
  - "problem" $\leftrightarrow$ "language",
  - "solve the problem" $\leftrightarrow$ " make a TM to accept the language"

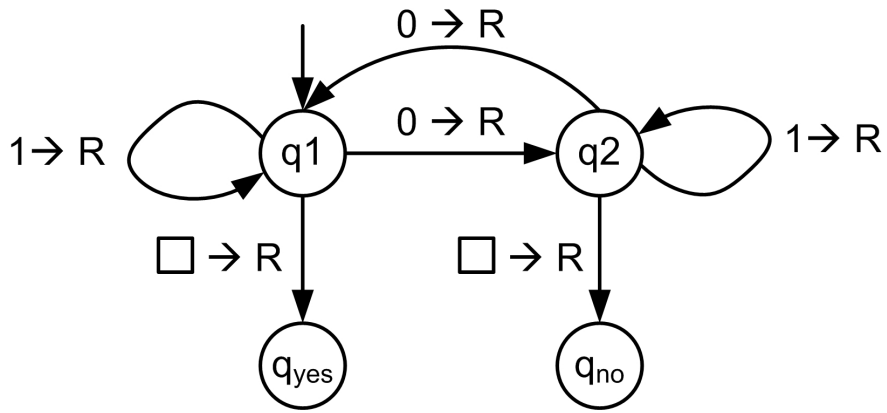
Turing Machine: Example

Example (1)

TM M_1 accepts the language $L = \{(1^m 01^p)^{2n} \mid m, n, p \geq 0\}$ over alphabet $\Sigma = \{0, 1\}$

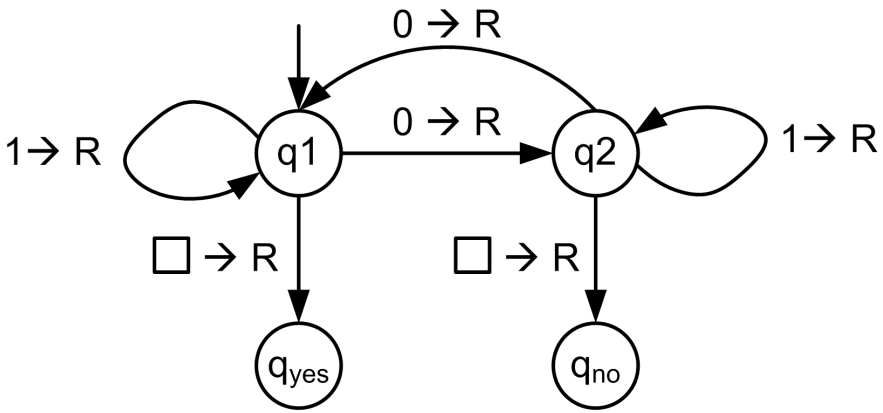
- $M_1 = \{Q, \Sigma, \Gamma, \delta, q_1, \{q_{yes}, q_{no}\}\}$
  - $Q = \{q_1, q_2, q_{yes}, q_{no}\}$
  - $\Sigma = \{0, 1\}$
  - $\Gamma = \{0, 1, \square\}$
  - δ defined in the following

$$\delta(q_1, 1) = (q_1, 1, R)$$
$$\delta(q_1, 0) = (q_2, 0, R)$$
$$\delta(q_1, \square) = (q_{yes}, \square, R)$$
$$\delta(q_2, 0) = (q_1, 0, R)$$
$$\delta(q_2, 1) = (q_2, 0, R)$$
$$\delta(q_2, \square) = (q_{no}, \square, R)$$



Turing Machine: Example

Given input: 1001



$$q_1 1001 \vdash 1 q_1 001 \vdash 10 q_2 01 \vdash 100 q_1 1 \vdash 1001 q_1 \square \vdash 1001 \square q_{yes}$$

Turing Machine: Exercise

Example (Exercise)

Describe a TM M_2 that accepts the language $L = \{0^{2^n} \mid n \geq 0\}$ over alphabet $\Sigma = \{0\}$

- $M_1 = \{Q, \Sigma, \Gamma, \delta, q_1, \{q_{yes}, q_{no}\}\}$
  - $Q = \{q_1, q_2, q_3, q_4, q_5, q_{yes}, q_{no}\}$
  - $\Sigma = \{0\}$
  - $\Gamma = \{0, x, \square\}$
  - $\delta = ?$

Turing Machine: The Universal Turing Machine

- Turing machine model is quite simple
- Can be easily simulated by a human
  - Provided enough pencils, tape space and patience
- Important result: *Machine can simulate machines*
  - Turing machines are *finite* objects!
  - Encoding into words over an alphabet
  - Also configurations are *finite*! Encode them too
- Simulator machine U only needs to
  - Receive an encode M as input
  - Input of M is w, give that also to U
  - U maintains encoded configurations of M and applies steps

Turing Machine: The Universal Turing Machine

- Let $\langle M \rangle$ be encoding of machine M

Theorem (The Universal Machine)

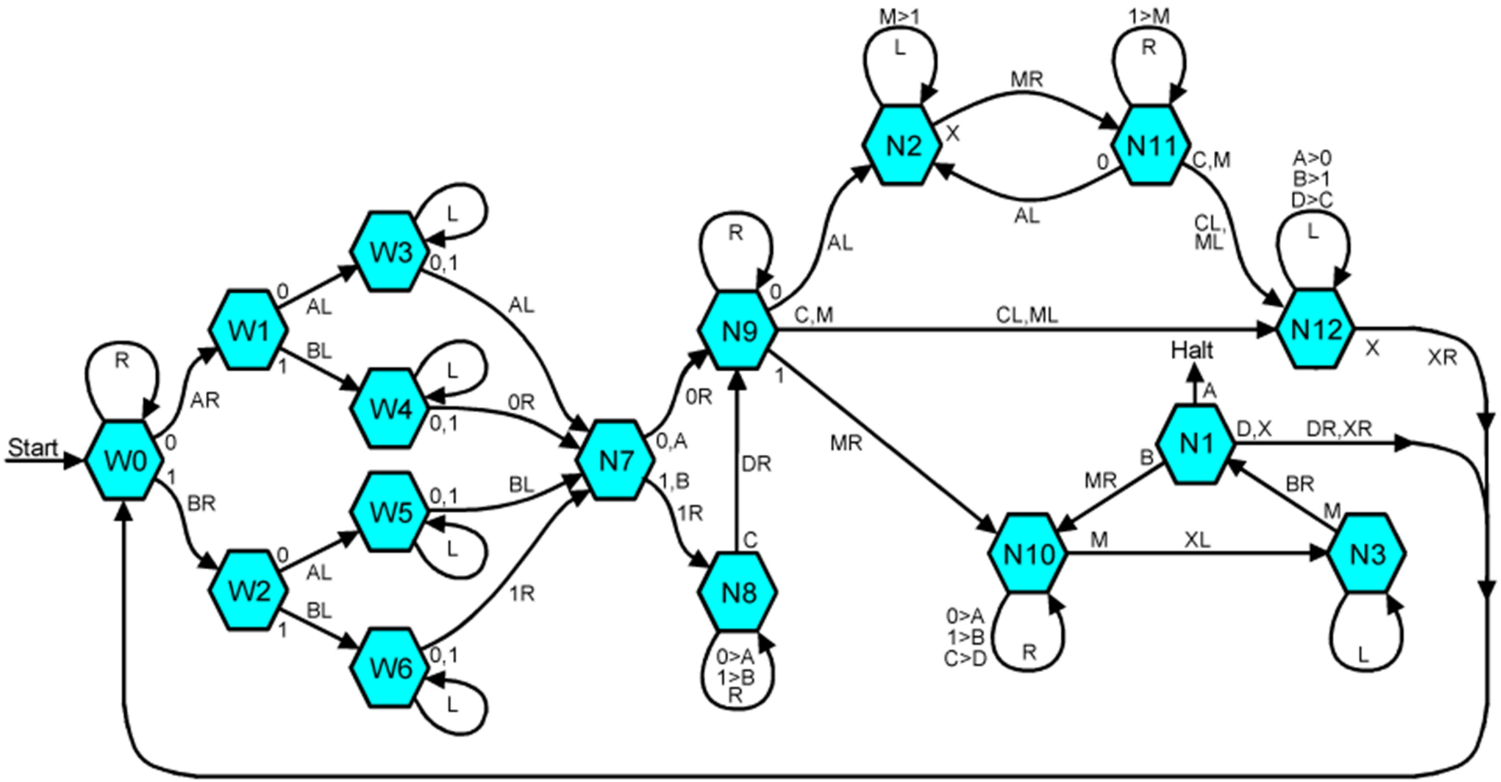
There exists a universal Turing Machine U, such that for all Turing machines M and all words $w \in \Sigma^*$:

$$U(\langle M \rangle, w) = M(w)$$

In particular, U doest not halt iff (if and only if) M does not halt.

(Without proof.)

Turing Machine: The Universal Turing Machine



Turing Machine: Multiple Tapes

- Definition so far: Machine uses *one* tape
- More convenient to have k tapes (k is a constant)
  - As dedicated input/output tapes
  - To save intermediate results
  - To precisely measure used space (except input/output space)
- Define this as *k-tape Turing Machines*
  - Still only one state, but k heads
  - Equivalent to 1-tape TM in terms of expressiveness (encode a "column" into one square)
  - Could be more efficient (roughly k times more efficient)

Turing Machine: Multiple Tapes

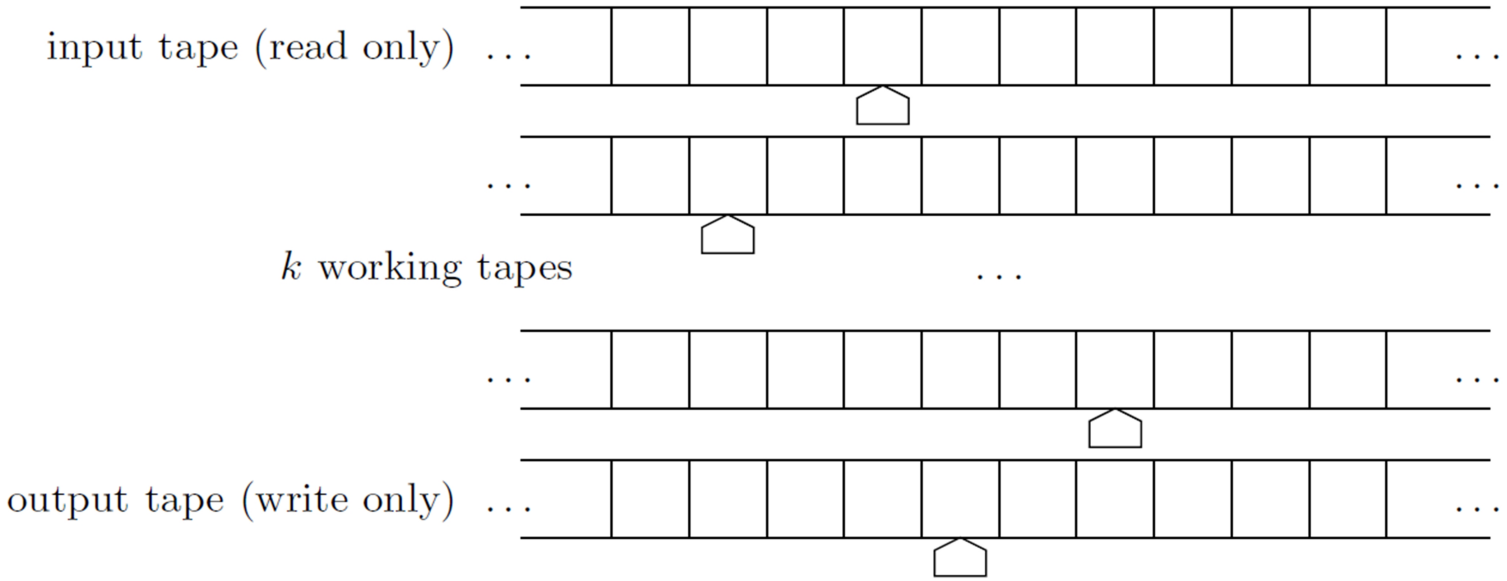


Figure: k-tape Turing Machine

Rest of the course: k-tape TM with dedicated input/output tapes

Turing Machine: Non-determinism

- Definition so far: Machine is *deterministic*
  - Exactly one next step possible
- Extension: allow different possible steps

$$\delta : (Q - F) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{R, N, L\})$$

- Machine chooses *non-deterministically* which step to do
  - Useful to model uncertainty in a system
  - Imagine behavior as a *computation tree*
  - Each path is one possible computation
  - Accepts w iff there is a path to q_{yes} (*accepting path*)
- Not a real machine, rather a theoretical model
- Will another characterization later
- Expressiveness does **not** increase in general (see following Theorem)

Turing Machine: Non-determinism

Theorem

Given a non-deterministic TM N, one can construct a deterministic TM M with $L(M) = L(N)$.
Further, if $N(w)$ accepts after $t(w)$ steps, then there is c such that $M(w)$ accepts after at most $c^{t(w)}$ steps.

- *Exponential* blowup concerning speed
- Ignoring speed, expressiveness is the same
- Note that N might not terminate on certain inputs

Turing Machine: Non-determinism

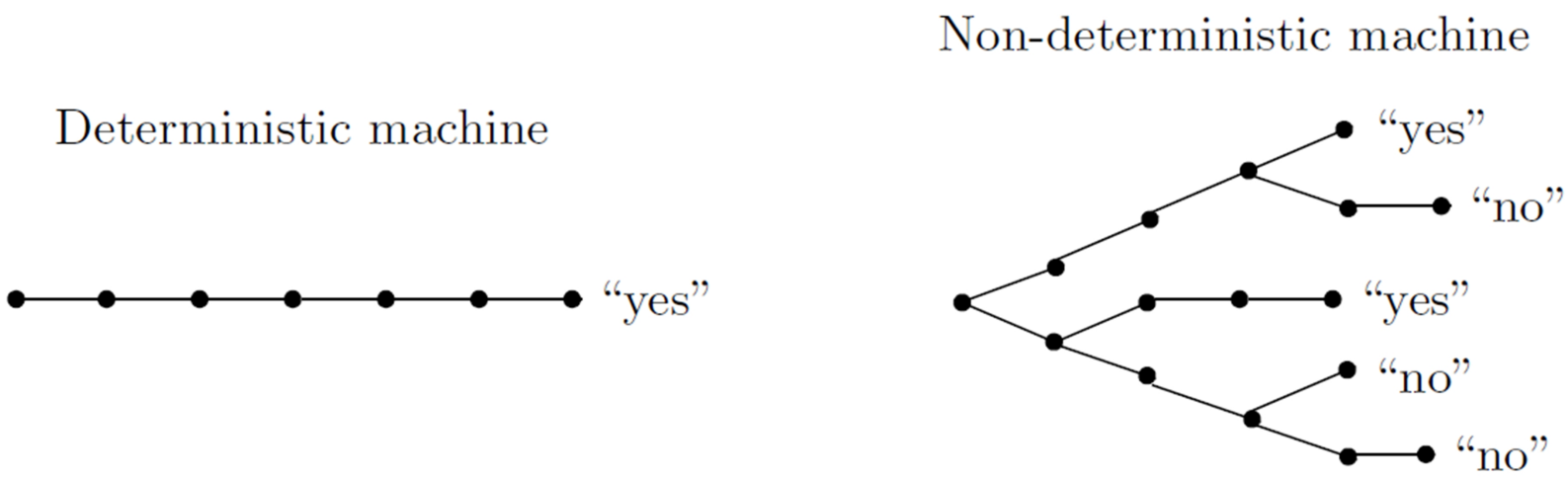


Figure: non-deterministic Turing Machine

Turing Machine: Summary

- *Simple* model of computation, but *powerful*
- Clearly defined syntax and semantics
- May *accept* languages or *compute* functions
- May use *multiple tapes*
- *Non-determinism* does not increase expressiveness
- A *Universal Machine* exists, simulating all other machines

- The machines we use from now on are
  - *deterministic*
  - *acceptors*
  - with k *tapes*

  (except explicitly stated otherwise)

Deciding a Problem

- Recall: Turing Machines running with input w may
  - halt in state q_{yes}
  - halt in state q_{no}
  - run *without halting*
- Given problem L and instance w, we want to *decide* whether $w \in L$:
  - using a machine M
  - if $w \in L$, M should halt in q_{yes}
  - if $w \notin L$, M should halt in q_{no}

Decidability and Undecidability

Definition

L is called *decidable*, if there exists a TM M with $L(M) = L$ that *halts on all inputs*. REC is the set of all decidable languages.

- We can *decide* the status of w by just running $M(w)$
- Termination guaranteed, we won't wait infinitely
- "M *decides* L"
- If $L \notin$ REC, then L is *undecidable*

Decidability and Undecidability: Example

Example (PRIMES $\in$ REC)

- Recall PRIMES $:= \{[p]_{10} \mid p \text{ is a prime number}\}$
- Can be decided:
  - Given $w = [p]_{10}$ for some n
  - Check for all $i \in (1, n)$ wehther n is multiple of i
  - If an i found: Halt in q_{no}
  - Otherwise, if all i negative answer: Halt in q_{yes}
- Can be implemented with a Turing Machine
- Always terminates (only finitely many i)
- Thus: PRIMES $\in$ REC

Semi-Decidability

Definition (Semi-Decidability)

L is called *semi-decidable*, if there exists a TM M with $L(M) = L$. RE is the set of all semi-decidable languages.

- Note the missing "*halts on all inputs*"!
- We can only "half-decide" the status of a given w:
  - Run M, wait for answer
  - If $w \in L$, M will halt in q_{yes}
  - If $w \notin L$, M may not halt
  - We don't know: $w \notin L$ or too impatient?

"M *semi-decides* L"

Class Differences

- Questions at this point:
  1. Are there undecidable problems?
  2. Can we at least semi-decide some of them?
  3. Are there any we cannot even semi-decide?
- Formally: $REC \subsetneq RE \subsetneq \mathcal{P}(\Sigma^*)$
- Difference between REC and RE: Termination guaranteed or not

Properties of Complementation

Theorem

1. $L \in REC \Leftrightarrow \overline{L} \in REC$. (*"closed under taking complements"*)
2. $L \in REC \Leftrightarrow (L \in RE \wedge \overline{L} \in RE)$

Proof (First part).

- Direction "$\Rightarrow$":
  - Assume M decides L and halts always
  - Construct M': Like M, but swap q_{yes} and q_{no}
  - M' decides $\overline{L}$ and always halts!
- Direction "$\Leftarrow$":
  - The same thing as above.

Properties of Complementation

Theorem

1. $L \in REC \Leftrightarrow \overline{L} \in REC$. (*"closed under taking complements"*)
2. $L \in REC \Leftrightarrow (L \in RE \wedge \overline{L} \in RE)$

Proof (Second part).

- Direction "$\Rightarrow$":
  - Follows from $REC \subset RE$ and first part
- Direction "$\Leftarrow$":
  - Let M_1, M_2 with $L(M_1) = L$ and $L(M_2) = \overline{L}$
  - Given w, simulate $M_1(w)$ and $M_2(w)$ step by step, in turns
  - *Eventually one of them will halt in q_{yes}*
  - If it was M_1, halt in q_{yes}
  - If it was M_2, halt in q_{no}
  - Thus, we always halt (and decide L)!

The Halting Problem

- Approach our three questions:
  1. Are there undecidable problems?
  2. Can we at least semi-decide some of them?
  3. Are there any we cannot even semi-decide?
- Classical problem: *Halting Problem*
  - Given a program M (Turing machine!) and an input w
  - *Will $M(w)$ terminate?*
  - Natural problem of great practical importance
- Formally: Let $\langle M \rangle$ be an encoding of M

Definition

H is the set of all Turing Machine encodings $\langle M \rangle$ and words w, such that *M halts on input w*:

$$H := \{(\langle M \rangle, w) \mid M(w) \neq \nearrow\}$$

Undecidability of the Halting Problem

Theorem

The halting problem is semi-decidable, but not decidable. Formally,

$$H \in RE - REC$$

Proof (First part).

We show $H \in RE$:

- Need to show: There is a TM M', such that
  - Given M and w
  - If $M(w)$ halts, M' accepts (halt in q_{yes})
  - If $M(w)$ does not halt, M' halts in q_{no} or does not halt
- Construct M': *Just simulate $M(w)$*
  - If simulation halts, accept (i.e., halt in q_{yes})
  - If simulation does not halt, we also won't
- Thus: $L(M') = H$

Undecidability of the Halting Problem

Theorem

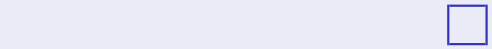
The halting problem is semi-decidable, but not decidable. Formally,

$$H \in RE - REC$$

Proof (Second part).

We show $H \notin REC$:

- Need to show: There is *no* TM M_H, such that
  - Given M and w
  - If $M(w)$ halts, M_H accepts (halt in q_{yes})
  - If $M(w)$ does not halt, M_H rejects (halts in q_{no})
- Simulation does not work!
  - What if it does not halt?
- New approach: Indirect proof
  - Above there exists such an M_H
  - Show a contradiction



Undecidability of the Halting Problem

Theorem

The halting problem is semi-decidable, but not decidable. Formally,

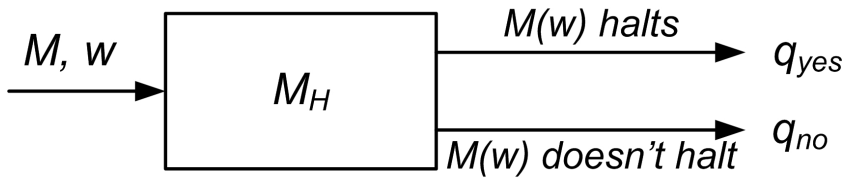
$$H \in RE - REC$$

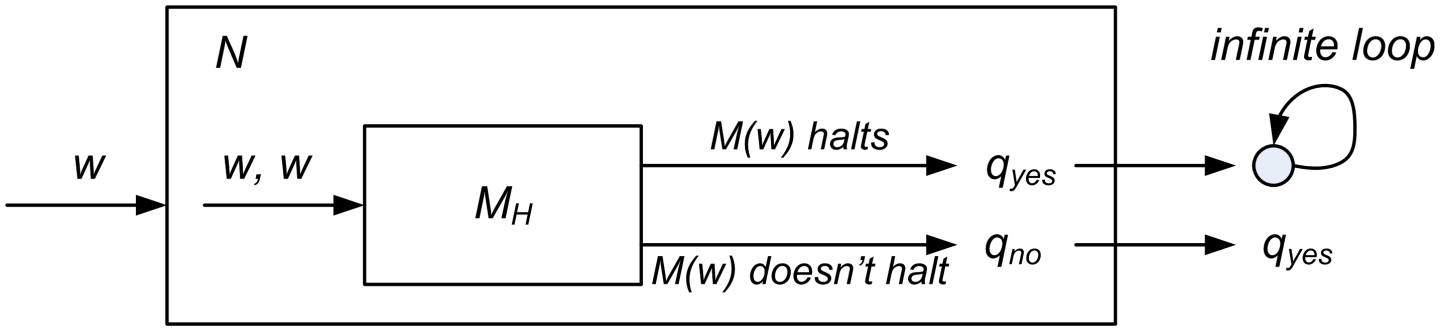
Proof (Second part, cont.)

We show $H \notin REC$:

- Assume there is M_H that solves the halting problem
  - $M_H(M, w)$ halts in q_{yes} if $M(w)$ halts
  - $M_H(M, w)$ halts in q_{no} if $M(w)$ does not
- Build another machine N:
  - On input w, run $M_H(w, w)$
    - the first w in $M_H(w, w)$ is the TM represented by w
  - If M_H halts in q_{yes}, enter infinite loop
  - If M_H halts in q_{no}, accept (i.e., halt in q_{yes})

Undecidability of the Halting Problem





Undecidability of the Halting Problem

Theorem

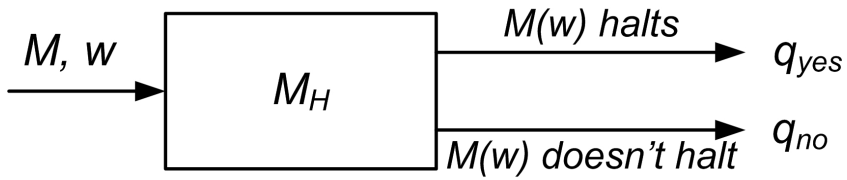
The halting problem is semi-decidable, but not decidable. Formally,

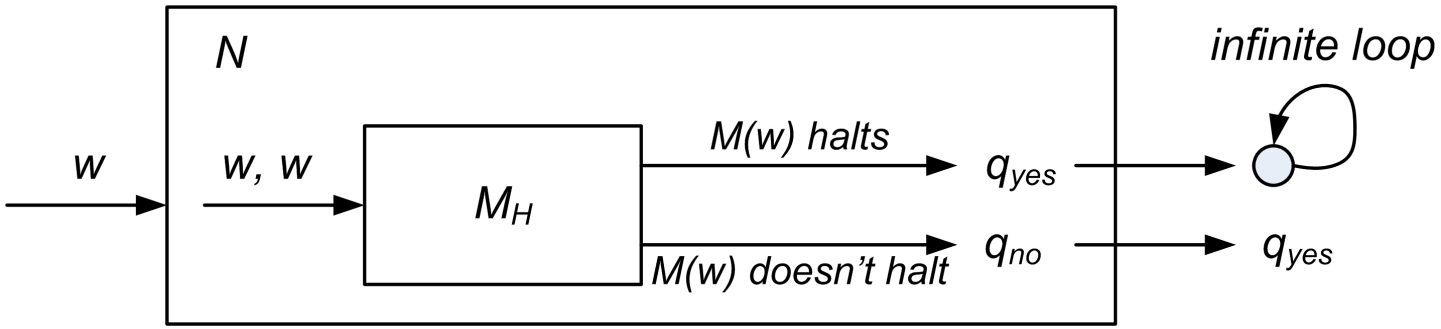
$$H \in RE - REC$$

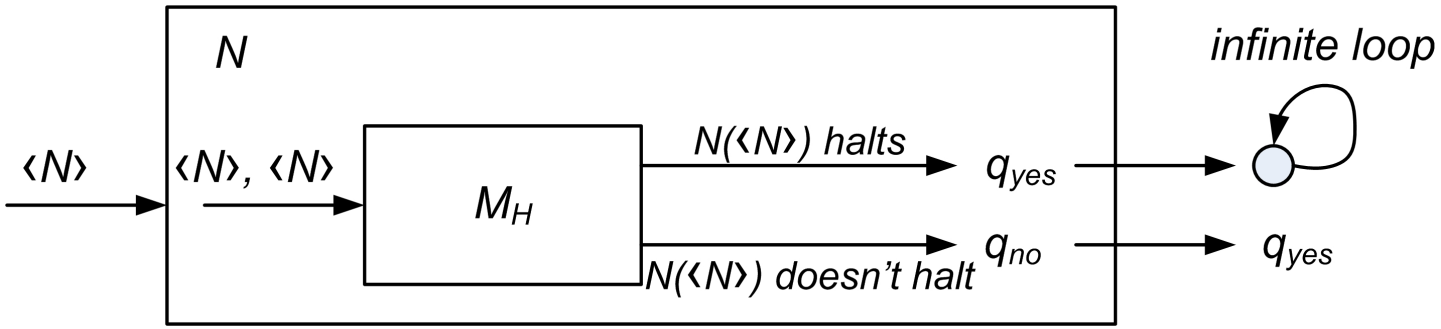
Proof (Second part, cont.)

- N is Turing Machine and $\langle N \rangle$ its coding. *Does $N(\langle N \rangle)$ halt?*
- Assume "yes, $N(\langle N \rangle)$ halts":
  - By construction of N, $M_H(\langle N \rangle, \langle N \rangle)$ halted in q_{no}
  - Definition of H: $N(\langle N \rangle)$ does not halt. *Contradiction!*
- Assume "no, $N(\langle N \rangle)$ does not halt":
  - By construction of N, $M_H(\langle N \rangle, \langle N \rangle)$ halted in q_{yes}
  - Definition of H: $N(\langle N \rangle)$ does halt. *Contradiction!*
- N cannot exist! $\Rightarrow$ M_H cannot exist.

Undecidability of the Halting Problem







Class Differences

- Know by now: $H \in RE - REC$, thus $REC \subsetneq RE$
- What about RE and $\mathcal{P}(\Sigma^*)$?
  - Is there an $L \subseteq \mathcal{P}(\Sigma^*)$ that is not semi-decidable?

Theorem

$$RE \subsetneq \mathcal{P}(\Sigma^*)$$

Proof.

- RE is *countable*
- $\mathcal{P}(\Sigma^*)$ is *uncountable*

□

But what is *countable* and *uncountable*?

Countable Set

Definition (Countable Set)

A countable set is a set with the same *cardinality* (number of elements) as some subset of $\mathbb{N}$.
A countable set is either a finite set or a *countably infinite* set

- What is cardinality $|A|$ of *infinite* sets A?
  - Two *infinite* sets has same cardinality if there is *one-to-one correspondence* between their elements
  - $|O| = |E|$
    - $O = \{1, 3, 5, \cdots\}$
    - $E = \{2, 4, 6, \cdots\}$
  - $|\mathbb{N}| = |E|$
    - $\mathbb{N} = \{1, 2, 3, \cdots\}$
    - $E = \{2, 4, 6, \cdots\}$
  - $|\mathbb{N}| = |\mathbb{R}|$ *not true!*

Countable Set

Theorem

$\mathbb{R}$ *is an uncountable set.*

Countable Set

We prove a simpler version of the theorem:

Theorem

The set of real numbers in $(0, 1)$ *is an uncountable set.*

Proof

- Let $f(n)$ be a total function maps a natural to a real
- We write $f(n)$ as a table:

n	$f(n)$									
1	0	.	3	1	4	1	5	9	2	...
2	0	.	3	7	3	7	3	7	3	...
3	0	.	1	4	2	8	5	7	1	...
4	0	.	7	0	7	1	0	6	7	...
5	0	.	3	7	5	0	0	0	0	...
⋮	⋮									

Countable Set

Proof.

- Let $f(n)$ be a total function maps a natural to a real
- We write $f(n)$ as a table:
- Construct a number x in $(0, 1)$, s.t.
  - its i^{th} digit differs the i^{th} digit of $f(i)$
- x must not be in the table, Proved.

Cantor's Diagonal Argument

n	$f(n)$									
1	0	.	3	1	4	1	5	9	2	...
2	0	.	3	7	3	7	3	7	3	...
3	0	.	1	4	2	8	5	7	1	...
4	0	.	7	0	7	1	0	6	7	...
5	0	.	3	7	5	0	0	0	0	...
⋮	⋮									

Cantor's Theorem

Theorem (Cantor's Theorem)

Given a (finite or infinite) set A and its power set $\mathcal{P}(A)$, it must hold

$$|A| < |\mathcal{P}(A)|$$

Proof.

Proved by Cantor's diagonal argument. □

Another Proof for Halting Problem

- Suppose a TM H exists that decides the halting problem,
- $H(\langle M \rangle, w) = 1$ means $M(w)$ halts, and 0 means $M(w)$ does not halt
- Construct D, s.t. D behaves the same as H, except that
  - $D(\langle D \rangle) = 1 - H(\langle H \rangle)$
- By construction, D is also a TM
- D cannot be in the table, thus D is not a TM, contradiction

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	...	$\langle D \rangle$	...
M_1	0	0	0	...	0	...
M_2	0	0	1	...	0	...
M_3	1	0	1	...	1	...
...	...	...	...	...	...	...
D	1	0	1	0	*???*	...
...	...	...	...	...	...	...

Rice's Theorem

- We know now: Halting Problem is undecidable
- Any other properties undecidable?
  - May be halting is just a strange property ...
- Actually, *all "non-trivial" behavioral properties are undecidable!*
  - Non-trivial: Some Turing Machines have it, some don't

Theorem (Rice's Theorem)

Let $\mathcal{C}$ be a non-trivial class of semi-decidable languages, i.e., $\emptyset \subsetneq \mathcal{C} \subsetneq RE$. The following language (problem) is undecidable:

$$L_{\mathcal{C}} := \{\langle M \rangle \mid L(\langle M \rangle) \in \mathcal{C}\}$$

(Without proof.)