Foundations of Computing

Lecture 18 – Exam Review

Arkady Yerukhimovich

March 25, 2025

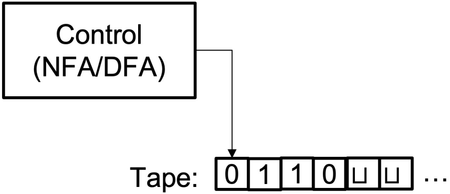
Outline

Lecture 17 Review

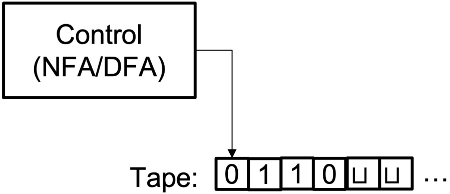
- Review of Reductions
- Types of Reductions – Mapping reductions, Turing reductions
- A brief intro into Kolmogorov complexity

Outline

The Turing Machine



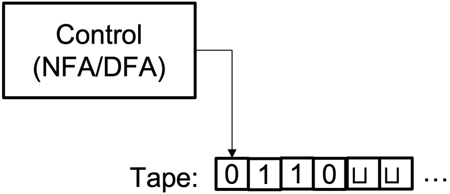
The Turing Machine



Key Differences:

- A TM can read and write to its tape

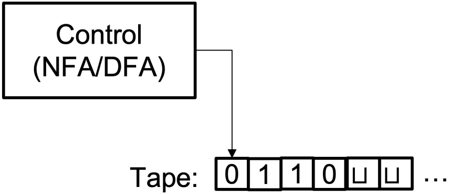
The Turing Machine



Key Differences:

- A TM can read and write to its tape
- The read/write head can move to the right and to the left

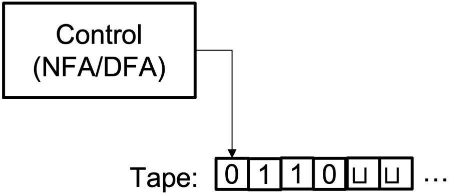
The Turing Machine



Key Differences:

- A TM can read and write to its tape
- The read/write head can move to the right and to the left
- No separate input tape, input written onto memory tape at start

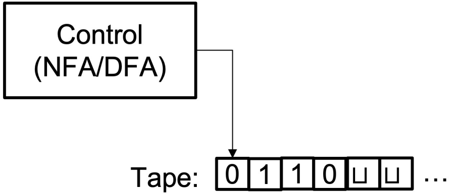
The Turing Machine



Key Differences:

- A TM can read and write to its tape
- The read/write head can move to the right and to the left
- No separate input tape, input written onto memory tape at start
- The memory tape is infinite

The Turing Machine



Key Differences:

- A TM can read and write to its tape
- The read/write head can move to the right and to the left
- No separate input tape, input written onto memory tape at start
- The memory tape is infinite
- Control FA has accept and reject states. If entered, TM halts and outputs.

An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

An Algorithm for M:
On input string s (written on the tape):

An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

An Algorithm for M:
On input string s (written on the tape):

1. Scan the input to check that it contains exactly one $\#$ symbol, if not reject.

An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

An Algorithm for M:
On input string s (written on the tape):

1. Scan the input to check that it contains exactly one $\#$ symbol, if not reject.
2. Zigzag to corresponding positions on each side of the $\#$ and see if they contain same symbol. If not, reject. Cross off symbols as they are checked

An Example: TM To Recognize $L = \{w\#w \mid w \in \{0,1\}^*\}$

An Algorithm for M:
On input string s (written on the tape):

1. Scan the input to check that it contains exactly one $\#$ symbol, if not reject.
2. Zigzag to corresponding positions on each side of the $\#$ and see if they contain same symbol. If not, reject. Cross off symbols as they are checked
3. When all symbols to the left of $\#$ have been crossed off, check that no uncrossed-off symbols remain to the right of $\#$. If any symbols remain, reject, otherwise accept.

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Observations:

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Observations:

- While unproven, all modern computers satisfy Church-Turing thesis

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Observations:

- While unproven, all modern computers satisfy Church-Turing thesis
- To prove that some problem cannot be solved by an algorithm, enough to reason about Turing Machines

Turing Machines and Algorithms

Church-Turing Thesis (1936)

Anything that can be computed by an algorithm can be computed by a Turing Machine

Observations:

- While unproven, all modern computers satisfy Church-Turing thesis
- To prove that some problem cannot be solved by an algorithm, enough to reason about Turing Machines
- This means that Turing Machines give an abstraction to capture "feasible computation"

Turing Machine – Formal Definition

A Turing machine M is a 7-tuple:

1. Q – set of states

Turing Machine – Formal Definition

A Turing machine M is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)

Turing Machine – Formal Definition

A Turing machine M is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$

Turing Machine – Formal Definition

A Turing machine M is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ – transition function

Turing Machine – Formal Definition

A Turing machine M is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ – transition function
5. $q_0 \in Q$ – start state
6. $q_{accept} \in Q$ – accept state
7. $q_{reject} \in Q$ – reject state

Initial State on input s:
M starts in state q_0 with $s\sqcup$ on the tape and tape head on s_0.

Transition function: $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

Turing Machine – Formal Definition

A Turing machine M is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ – transition function
5. $q_0 \in Q$ – start state
6. $q_{accept} \in Q$ – accept state
7. $q_{reject} \in Q$ – reject state

Initial State on input s:
M starts in state q_0 with $s\sqcup$ on the tape and tape head on s_0.

Transition function: $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
On state q and tape input γ:

Turing Machine – Formal Definition

A Turing machine M is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ – transition function
5. $q_0 \in Q$ – start state
6. $q_{accept} \in Q$ – accept state
7. $q_{reject} \in Q$ – reject state

Initial State on input s:
M starts in state q_0 with $s\sqcup$ on the tape and tape head on s_0.

Transition function: $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
On state q and tape input γ:

- move control to state q',

Turing Machine – Formal Definition

A Turing machine M is a 7-tuple:

1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ – transition function
5. $q_0 \in Q$ – start state
6. $q_{accept} \in Q$ – accept state
7. $q_{reject} \in Q$ – reject state

Initial State on input s:
M starts in state q_0 with $s\sqcup$ on the tape and tape head on s_0.

Transition function: $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
On state q and tape input γ:

- move control to state q',
- write γ' to the tape,

Turing Machine – Formal Definition

A Turing machine M is a 7-tuple:

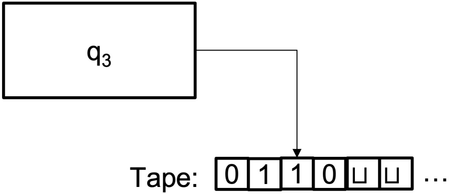
1. Q – set of states
2. Σ – input alphabet (not including blank symbol $\sqcup$)
3. Γ – tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ – transition function
5. $q_0 \in Q$ – start state
6. $q_{accept} \in Q$ – accept state
7. $q_{reject} \in Q$ – reject state

Initial State on input s:
M starts in state q_0 with $s\sqcup$ on the tape and tape head on s_0.

Transition function: $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
On state q and tape input γ:

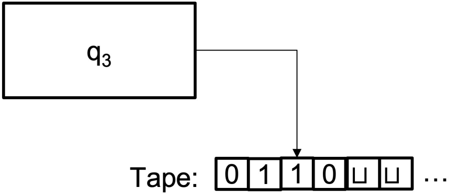
- move control to state q',
- write γ' to the tape,
- and move the tape head one spot to either Left or Right

Computing on a Turing Machine



Configuration of a TM

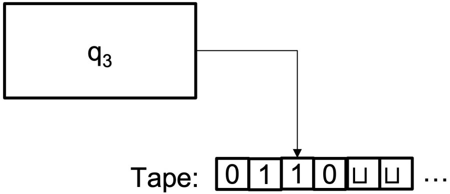
Computing on a Turing Machine



Configuration of a TM

- Describes the state of a TM computation

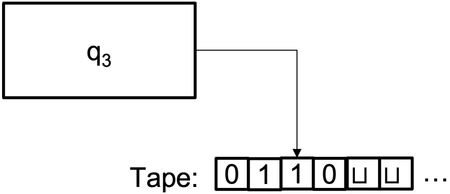
Computing on a Turing Machine



Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head

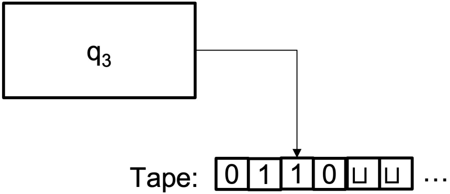
Computing on a Turing Machine



Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$

Computing on a Turing Machine



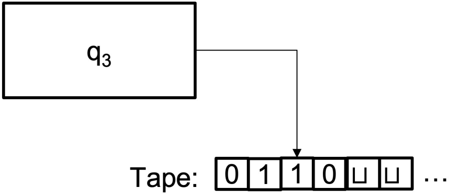
Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$

Definitions:

- Configuration C_1 **yields** C_2, if M can go from C_1 to C_2 in a single step

Computing on a Turing Machine



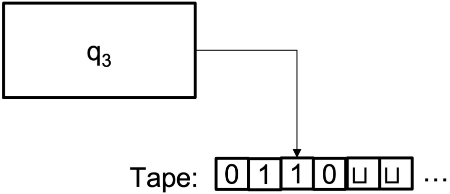
Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$

Definitions:

- Configuration C_1 **yields** C_2, if M can go from C_1 to C_2 in a single step
- start configuration of M on input s – configuration q_0s

Computing on a Turing Machine



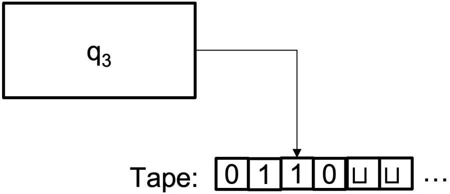
Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$

Definitions:

- Configuration C_1 **yields** C_2, if M can go from C_1 to C_2 in a single step
- start configuration of M on input s – configuration q_0s
- accepting configuration – any config with state q_{accept}

Computing on a Turing Machine



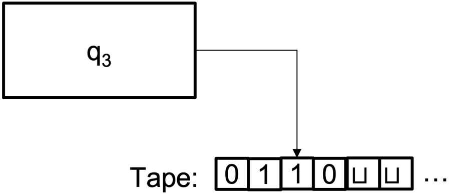
Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$

Definitions:

- Configuration C_1 **yields** C_2, if M can go from C_1 to C_2 in a single step
- start configuration of M on input s – configuration q_0s
- accepting configuration – any config with state q_{accept}
- rejecting configuration – any config with state q_{reject}

Computing on a Turing Machine



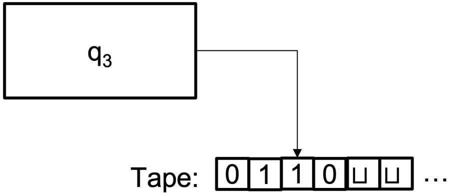
Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$

Definitions:

- Configuration C_1 **yields** C_2, if M can go from C_1 to C_2 in a single step
- start configuration of M on input s – configuration q_0s
- accepting configuration – any config with state q_{accept}
- rejecting configuration – any config with state q_{reject}
- halting configuration – accepting or rejecting configs

Computing on a Turing Machine



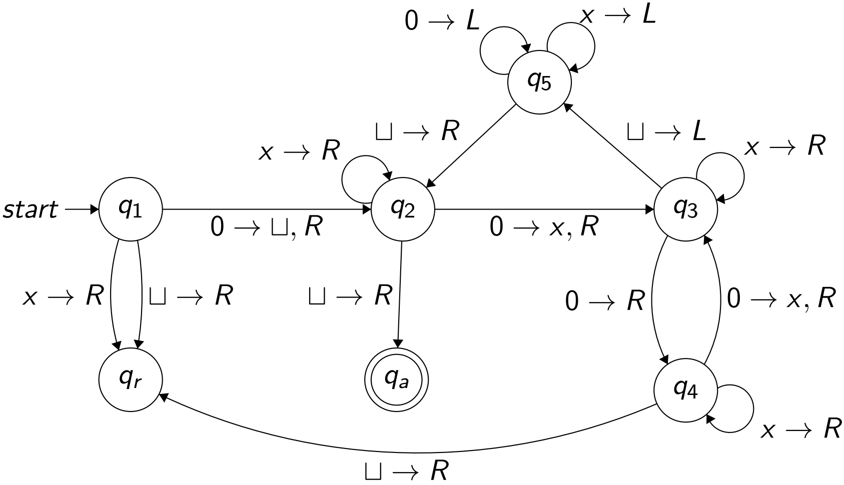
Configuration of a TM

- Describes the state of a TM computation
- Current state of control, state of tape, location of tape head
- Example: $01q_310$

Definitions:

- Configuration C_1 **yields** C_2, if M can go from C_1 to C_2 in a single step
- start configuration of M on input s – configuration q_0s
- accepting configuration – any config with state q_{accept}
- rejecting configuration – any config with state q_{reject}
- halting configuration – accepting or rejecting configs

Full Specification: Running M on $w = 0000$



Outline

Characterizing Computability of Languages

Definition: Recursively enumerable languages

Characterizing Computability of Languages

Definition: Recursively enumerable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

Characterizing Computability of Languages

Definition: Recursively enumerable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

- M halts and accepts all strings in L

Characterizing Computability of Languages

Definition: Recursively enumerable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

- M halts and accepts all strings in L
- M may not halt on strings not in L – does not necessarily have to reject

Characterizing Computability of Languages

Definition: Recursively enumerable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

- M halts and accepts all strings in L
- M may not halt on strings not in L – does not necessarily have to reject

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

Characterizing Computability of Languages

Definition: Recursively enumerable languages

A language L is *Turing-recognizable* or *recursively enumerable* if some TM M recognizes it

- M halts and accepts all strings in L
- M may not halt on strings not in L – does not necessarily have to reject

Definition: Decidable languages

A language L is *decidable* or *recursive* if some TM M decides it

- M halts on all inputs, accepting those in L and rejecting those not in L

Take Away

You should be able to show that a language is decidable or Turing-recognizable by designing a TM algorithm.

Important TM Notation / Observations

- TM always takes a string as input
  - Sometimes we want to talk about a TM taking another type of input (e.g., a graph, a FA, a TM)
  - To do so, we must serialize the object into a string
  - Notation: $\langle G \rangle$

Important TM Notation / Observations

- TM always takes a string as input
  - Sometimes we want to talk about a TM taking another type of input (e.g., a graph, a FA, a TM)
  - To do so, we must serialize the object into a string
  - Notation: $\langle G \rangle$
- We can "mark" cells on the tape
  - Notation: $\dot{x}$
  - Technically, this is adding a symbol to Γ

Important TM Notation / Observations

- TM always takes a string as input
  - Sometimes we want to talk about a TM taking another type of input (e.g., a graph, a FA, a TM)
  - To do so, we must serialize the object into a string
  - Notation: $\langle G \rangle$
- We can "mark" cells on the tape
  - Notation: $\dot{x}$
  - Technically, this is adding a symbol to Γ
- Can use multiple tapes if it's useful

Important TM Notation / Observations

- TM always takes a string as input
  - Sometimes we want to talk about a TM taking another type of input (e.g., a graph, a FA, a TM)
  - To do so, we must serialize the object into a string
  - Notation: $\langle G \rangle$
- We can "mark" cells on the tape
  - Notation: $\dot{x}$
  - Technically, this is adding a symbol to Γ
- Can use multiple tapes if it's useful
- Can give a machine as an input to another machine
  - All machines we have seen can be written as finite tuples, e.g. $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$
  - So, we can write this as a string and pass it to a TM
  - TM can then run the machine from this description

Important TM Notation / Observations

- TM always takes a string as input
  - Sometimes we want to talk about a TM taking another type of input (e.g., a graph, a FA, a TM)
  - To do so, we must serialize the object into a string
  - Notation: $\langle G \rangle$
- We can "mark" cells on the tape
  - Notation: $\dot{x}$
  - Technically, this is adding a symbol to Γ
- Can use multiple tapes if it's useful
- Can give a machine as an input to another machine
  - All machines we have seen can be written as finite tuples, e.g. $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$
  - So, we can write this as a string and pass it to a TM
  - TM can then run the machine from this description
  - A TM that accepts any TM and runs it is called a *universal TM*

Specification of a Turing Machine

There are several levels of detail for specifying a TM

1. Full specification
   - Give full detail of transition function δ
   - This is very tedious

Specification of a Turing Machine

There are several levels of detail for specifying a TM

1. Full specification
   - Give full detail of transition function δ
   - This is very tedious
2. Turing Machine Algorithm specification
   - Explain algorithmically what happens on the tape
   - For example, scan the tape until you find a $\#$, zig-zag on the tape, etc.
   - Don't bother specifying a DFA for the control state

Specification of a Turing Machine

There are several levels of detail for specifying a TM

1. Full specification
    - Give full detail of transition function δ
    - This is very tedious
2. Turing Machine Algorithm specification
    - Explain algorithmically what happens on the tape
    - For example, scan the tape until you find a $\#$, zig-zag on the tape, etc.
    - Don't bother specifying a DFA for the control state
3. Algorithm specification
    - Give algorithm in pseudocode
    - Don't explicitly spell out what happens on the tape

Turing Machine Variants

- Multi-tape Turing Machine
- Nondeterministic Turing Machine

What You Need to Know

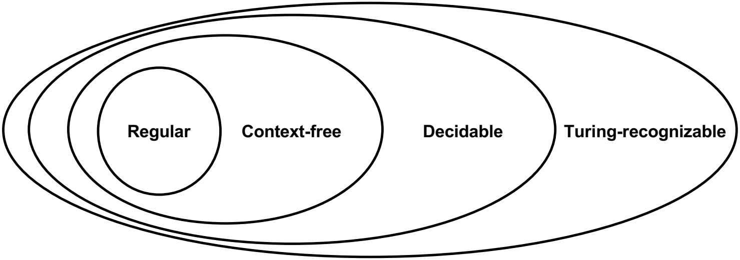
- Be able to explain what the variant is
- Know whether it is equivalent to standard TM
- Be able to explain why

Decidable Languages

We have seen many examples of decidable languages:

- Languages about strings
- Languages about DFAs/NFAs/CFGs – know which ones are decidable and which are not, why
- Be comfortable with TM's that take another machine as input

Relationships Among Language Classes



Outline

Intuition: Countable sets are ones where we can arrange elements into a "first element", "second element", and so on.

Preliminaries – Countable and Uncountable Sets

Intuition: Countable sets are ones where we can arrange elements into a "first element", "second element", and so on.

- An infinite set A is *countably infinite* if it has the same cardinality as the natural numbers: $\mathcal{N} = 1, 2, 3, \ldots$

Preliminaries – Countable and Uncountable Sets

Intuition: Countable sets are ones where we can arrange elements into a "first element", "second element", and so on.

- An infinite set A is *countably infinite* if it has the same cardinality as the natural numbers: $\mathcal{N} = 1, 2, 3, \ldots$
- A set A is countable if it is finite or countably infinite

Preliminaries – Countable and Uncountable Sets

Intuition: Countable sets are ones where we can arrange elements into a "first element", "second element", and so on.

- An infinite set A is *countably infinite* if it has the same cardinality as the natural numbers: $\mathcal{N} = 1, 2, 3, \ldots$
- A set A is countable if it is finite or countably infinite
- A set that is not countable is *uncountable*

Diagonalization

Real Numbers

The set of real numbers ($\mathcal{R}$) is uncountable

Proof: By diagonalization

Diagonalization

Real Numbers

The set of real numbers ($\mathcal{R}$) is uncountable

Proof: By diagonalization

- Assume that $\mathcal{R}$ is countable

Diagonalization

Real Numbers

The set of real numbers ($\mathcal{R}$) is uncountable

Proof: By diagonalization

- Assume that $\mathcal{R}$ is countable
- Then there is a one-to-one and onto mapping f from $\mathcal{N}$ to $\mathcal{R}$

Diagonalization

Real Numbers

The set of real numbers ($\mathcal{R}$) is uncountable

Proof: By diagonalization

- Assume that $\mathcal{R}$ is countable
- Then there is a one-to-one and onto mapping f from $\mathcal{N}$ to $\mathcal{R}$

n	f(n)
1	1.234...
2	3.141...
3	5.556...
$\vdots$	$\vdots$

Diagonalization

Real Numbers

The set of real numbers ($\mathcal{R}$) is uncountable

Proof: By diagonalization

- Assume that $\mathcal{R}$ is countable
- Then there is a one-to-one and onto mapping f from $\mathcal{N}$ to $\mathcal{R}$

n	f(n)
1	1.234...
2	3.141...
3	5.556...
$\vdots$	$\vdots$

- We construct a value $x \in \mathcal{R}$ s.t $x \neq f(n)$ for any n
  Idea: For all $i \in \mathcal{N}$, make $x_i \neq f(i)_i$

Diagonalization

Real Numbers

The set of real numbers ($\mathcal{R}$) is uncountable

Proof: By diagonalization

- Assume that $\mathcal{R}$ is countable
- Then there is a one-to-one and onto mapping f from $\mathcal{N}$ to $\mathcal{R}$

n	f(n)
1	1.234...
2	3.141...
3	5.556...
⋮	⋮

- We construct a value $x \in \mathcal{R}$ s.t $x \neq f(n)$ for any n
  Idea: For all $i \in \mathcal{N}$, make $x_i \neq f(i)_i$
- Contradiction – f is not mapping between $\mathcal{R}$ and $\mathcal{N}$

A_{TM} is Turing-recognizable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

A_{TM} is Turing-recognizable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By construction of machine $M_{A_{TM}}$

$M_{A_{TM}}$: On input $\langle M, w\rangle$,

1. Run M on input w

A_{TM} is Turing-recognizable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By construction of machine $M_{A_{TM}}$

$M_{A_{TM}}$: On input $\langle M, w\rangle$,

1. Run M on input w
2. If M halts, halt and output what M outputs

A_{TM} is Turing-recognizable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By construction of machine $M_{A_{TM}}$

$M_{A_{TM}}$: On input $\langle M, w\rangle$,

1. Run M on input w
2. If M halts, halt and output what M outputs

Correctness:

- For any input $\langle M, w\rangle \in A_{TM}$, M is a TM, and $M(w)$ halts and outputs 1.

A_{TM} is Turing-recognizable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By construction of machine $M_{A_{TM}}$

$M_{A_{TM}}$: On input $\langle M, w\rangle$,

1. Run M on input w
2. If M halts, halt and output what M outputs

Correctness:

- For any input $\langle M, w\rangle \in A_{TM}$, M is a TM, and $M(w)$ halts and outputs 1.
- Hence, $M_{A_{TM}}$, also halts and outputs 1

A_{TM} is Turing-recognizable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By construction of machine $M_{A_{TM}}$

$M_{A_{TM}}$: On input $\langle M, w\rangle$,

1. Run M on input w
2. If M halts, halt and output what M outputs

Correctness:

- For any input $\langle M, w\rangle \in A_{TM}$, M is a TM, and $M(w)$ halts and outputs 1.
- Hence, $M_{A_{TM}}$, also halts and outputs 1
- Thus, $M_{A_{TM}}$ accepts all inputs in A_{TM}

A_{TM} is Turing-recognizable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By construction of machine $M_{A_{TM}}$

$M_{A_{TM}}$: On input $\langle M, w\rangle$,

1. Run M on input w
2. If M halts, halt and output what M outputs

Correctness:

- For any input $\langle M, w\rangle \in A_{TM}$, M is a TM, and $M(w)$ halts and outputs 1.
- Hence, $M_{A_{TM}}$, also halts and outputs 1
- Thus, $M_{A_{TM}}$ accepts all inputs in A_{TM}
- Note that $M_{A_{TM}}$ may not halt on all inputs – doesn't decide A_{TM}

A_{TM} is Undecidable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

A_{TM} is Undecidable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By contradiction

A_{TM} is Undecidable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By contradiction

- Assume that A_{TM} is decided by TM H

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

A_{TM} is Undecidable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By contradiction

- Assume that A_{TM} is decided by TM H

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

- Use H to build a TM D that checks whether a TM M accepts its own description, and then does the opposite:

A_{TM} is Undecidable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By contradiction

- Assume that A_{TM} is decided by TM H

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

- Use H to build a TM D that checks whether a TM M accepts its own description, and then does the opposite:
  On Input $\langle M\rangle$, where M is a TM
  1. Run H on input $\langle M, \langle M\rangle\rangle$
  2. Output the opposite of what H outputs

A_{TM} is Undecidable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By contradiction

- Assume that A_{TM} is decided by TM H

$$H(\langle M, w\rangle) = \begin{cases} accept & \text{if } M \text{ accepts } w \\ reject & \text{if } M \text{ does not accept } w \end{cases}$$

- Use H to build a TM D that checks whether a TM M accepts its own description, and then does the opposite:
  On Input $\langle M\rangle$, where M is a TM
  1. Run H on input $\langle M, \langle M\rangle\rangle$
  2. Output the opposite of what H outputs

$$D(\langle M\rangle) = \begin{cases} accept & \text{if } M \text{ does not accept } \langle M\rangle \\ reject & \text{if } M \text{ accepts } \langle M\rangle \end{cases}$$

A_{TM} is Undecidable

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M(w) = 1\}$$

Proof: By contradiction

- Assume that A_{TM} is decided by TM H

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

- Use H to build a TM D that checks whether a TM M accepts its own description, and then does the opposite:
  On Input $\langle M\rangle$, where M is a TM
  1. Run H on input $\langle M, \langle M\rangle\rangle$
  2. Output the opposite of what H outputs

$$D(\langle M\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M\rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M\rangle \end{cases}$$

- Now consider what happens if we run D on $\langle D\rangle$

$$D(\langle D\rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D\rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D\rangle \end{cases}$$

How Is This a Diagonalization?

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	$\underline{accept}$	$reject$	$accept$		$accept$	
M_2	$reject$	$\underline{reject}$	$reject$	$\ldots$	$accept$	$\ldots$
M_3	$accept$	$accept$	$\underline{accept}$		$reject$	
$\vdots$		$\vdots$		$\ddots$		
D	$reject$	$accept$	$reject$		?	

- We have defined D to do the opposite of what M_i does on input $\langle M_i \rangle$
- But what does D do on input $\langle D \rangle$??

Outline

Main Observation

Suppose that $A \leq B$, then:

Reductions and Undecidability

Main Observation

Suppose that $A \leq B$, then:

- If A is undecidable
- B must also be undecidable

Reductions and Undecidability

Main Observation

Suppose that $A \leq B$, then:

- If A is undecidable
- B must also be undecidable

Proof: (by contradiction)

Reductions and Undecidability

Main Observation

Suppose that $A \leq B$, then:

- If A is undecidable
- B must also be undecidable

Proof: (by contradiction)

- Suppose that B is decidable

Main Observation

Suppose that $A \leq B$, then:

- If A is undecidable
- B must also be undecidable

Proof: (by contradiction)

- Suppose that B is decidable
- Since $A \leq B$, there exists an algorithm (i.e., a reduction) that uses a solution to B to solve A

Reductions and Undecidability

Main Observation

Suppose that $A \leq B$, then:

- If A is undecidable
- B must also be undecidable

Proof: (by contradiction)

- Suppose that B is decidable
- Since $A \leq B$, there exists an algorithm (i.e., a reduction) that uses a solution to B to solve A
- But, this means that A is decidable by running the machine for B as needed by the reduction

Undecidability of HALT

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Undecidability of HALT

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem: $HALT$ is undecidable

Undecidability of HALT

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem: $HALT$ is undecidable

Proof Sketch:

- We show that $A_{TM} \leq HALT$

Undecidability of HALT

$$HALT_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem: $HALT$ is undecidable

Proof Sketch:

- We show that $A_{TM} \leq HALT$
- Since we know that A_{TM} is undecidable, this shows that $HALT$ is also undecidable

Undecidability of HALT

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem: $HALT$ is undecidable

Proof Sketch:

- We show that $A_{TM} \leq HALT$
- Since we know that A_{TM} is undecidable, this shows that $HALT$ is also undecidable

Proof:

Construct reduction R that decides A_{TM} given a TM D that decides $HALT$

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem: $HALT$ is undecidable

Proof Sketch:

- We show that $A_{TM} \leq HALT$
- Since we know that A_{TM} is undecidable, this shows that $HALT$ is also undecidable

Proof:

Construct reduction R that decides A_{TM} given a TM D that decides $HALT$

On input $\langle M, w\rangle$, R does the following:

Undecidability of HALT

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem: $HALT$ is undecidable

Proof Sketch:

- We show that $A_{TM} \leq HALT$
- Since we know that A_{TM} is undecidable, this shows that $HALT$ is also undecidable

Proof:

Construct reduction R that decides A_{TM} given a TM D that decides $HALT$

On input $\langle M, w\rangle$, R does the following:

- Run $D(\langle M, w\rangle)$

Undecidability of HALT

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem: $HALT$ is undecidable

Proof Sketch:

- We show that $A_{TM} \leq HALT$
- Since we know that A_{TM} is undecidable, this shows that $HALT$ is also undecidable

Proof:

Construct reduction R that decides A_{TM} given a TM D that decides $HALT$

On input $\langle M, w\rangle$, R does the following:

- Run $D(\langle M, w\rangle)$
- If D rejects – $M(w)$ doesn't halt – halt and reject

Undecidability of HALT

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem: $HALT$ is undecidable

Proof Sketch:

- We show that $A_{TM} \leq HALT$
- Since we know that A_{TM} is undecidable, this shows that $HALT$ is also undecidable

Proof:

Construct reduction R that decides A_{TM} given a TM D that decides $HALT$

On input $\langle M, w\rangle$, R does the following:

- Run $D(\langle M, w\rangle)$
- If D rejects – $M(w)$ doesn't halt – halt and reject
- if D accepts – $M(w)$ halts – Simulate $M(w)$ until it halts

Undecidability of HALT

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem: $HALT$ is undecidable

Proof Sketch:

- We show that $A_{TM} \leq HALT$
- Since we know that A_{TM} is undecidable, this shows that $HALT$ is also undecidable

Proof:

Construct reduction R that decides A_{TM} given a TM D that decides $HALT$

On input $\langle M, w\rangle$, R does the following:

- Run $D(\langle M, w\rangle)$
- If D rejects – $M(w)$ doesn't halt – halt and reject
- if D accepts – $M(w)$ halts – Simulate $M(w)$ until it halts
- Output whatever M output

Importance of Algorithms

Algorithms

Algorithms are critical for understanding decidability of problems

Importance of Algorithms

Algorithms

Algorithms are critical for understanding decidability of problems

1. To show that a problem is decidable – give an algorithm that always terminates and outputs the answer

Importance of Algorithms

Algorithms

Algorithms are critical for understanding decidability of problems

1. To show that a problem is decidable – give an algorithm that always terminates and outputs the answer
2. To show that a problem is undecidable – give an algorithm (a reduction) that shows that this problem can be used to solve one of the undecidable problems

What You Need to Know

You should be able to:

- Understand which direction a reduction should go

What You Need to Know

You should be able to:

- Understand which direction a reduction should go
- Understand implications of such a reduction

What You Need to Know

You should be able to:

- Understand which direction a reduction should go
- Understand implications of such a reduction
- Give a reduction between two related languages

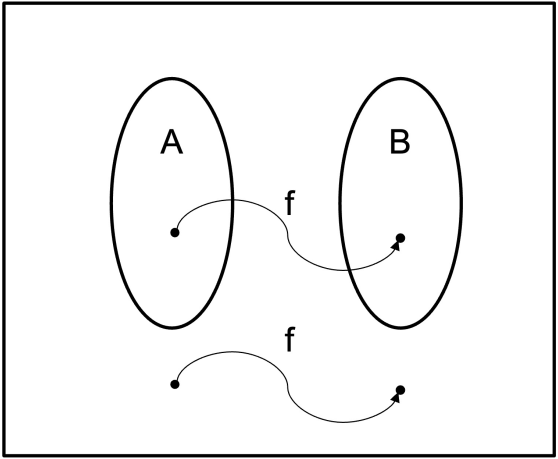
Reduction Types

Know the difference between:

- Mapping reductions
- Turing reductions

Know what each one implies

Mapping Reductions



Mapping Reduction Properties

Mapping reductions are very useful:

Mapping Reduction Properties

Mapping reductions are very useful:

1. If $A \leq_m B$
   - If B is decidable then A is decidable

Mapping Reduction Properties

Mapping reductions are very useful:

1. If $A \leq_m B$
   - If B is decidable then A is decidable
   - If A is undecidable then B is undecidable

Mapping Reduction Properties

Mapping reductions are very useful:

1. If $A \leq_m B$
   - If B is decidable then A is decidable
   - If A is undecidable then B is undecidable
2. If $A \leq_m B$
   - If B is Turing-recognizable then

Mapping Reduction Properties

Mapping reductions are very useful:

1. If $A \leq_m B$
   - If B is decidable then A is decidable
   - If A is undecidable then B is undecidable
2. If $A \leq_m B$
   - If B is Turing-recognizable then A is Turing-recognizable

Mapping Reduction Properties

Mapping reductions are very useful:

1. If $A \leq_m B$
   - If B is decidable then A is decidable
   - If A is undecidable then B is undecidable
2. If $A \leq_m B$
   - If B is Turing-recognizable then A is Turing-recognizable
   - If A is not Turing-recognizable than B is not Turing-recognizable

Turing Reductions

Definition

Language A is Turing reducible to language B ($A \leq_T B$) if can use a decider for B to decide A.

Turing Reductions

Definition

Language A is Turing reducible to language B ($A \leq_T B$) if can use a decider for B to decide A.

- The reduction may make multiple calls to decider for B and may not directly use the result.

Turing Reduction Properties

Turing reductions are more general than mapping reductions:

Turing Reduction Properties

Turing reductions are more general than mapping reductions:

1. If $A \leq_m B$, then $A \leq_T B$

Turing Reduction Properties

Turing reductions are more general than mapping reductions:

1. If $A \leq_m B$, then $A \leq_T B$
2. If $A \leq_T B$, then it is not necessarily the case that $A \leq_m B$

Turing Reduction Properties

Turing reductions are more general than mapping reductions:

1. If $A \leq_m B$, then $A \leq_T B$
2. If $A \leq_T B$, then it is not necessarily the case that $A \leq_m B$
    - In particular, $L_{TM} \leq_T \overline{L_{TM}}$, but $L_{TM} \not\leq_m \overline{L_{TM}}$

Turing Reduction Properties

Turing reductions are more general than mapping reductions:

1. If $A \leq_m B$, then $A \leq_T B$
2. If $A \leq_T B$, then it is not necessarily the case that $A \leq_m B$
   - In particular, $L_{TM} \leq_T \overline{L_{TM}}$, but $L_{TM} \not\leq_m \overline{L_{TM}}$

But, they have weaker implications than mapping reductions:

Turing Reduction Properties

Turing reductions are more general than mapping reductions:

1. If $A \leq_m B$, then $A \leq_T B$
2. If $A \leq_T B$, then it is not necessarily the case that $A \leq_m B$
   - In particular, $L_{TM} \leq_T \overline{L_{TM}}$, but $L_{TM} \not\leq_m \overline{L_{TM}}$

But, they have weaker implications than mapping reductions:

3. If $A \leq_T B$
   - If B is decidable then A is decidable

Turing Reduction Properties

Turing reductions are more general than mapping reductions:

1. If $A \leq_m B$, then $A \leq_T B$
2. If $A \leq_T B$, then it is not necessarily the case that $A \leq_m B$
   - In particular, $L_{TM} \leq_T \overline{L_{TM}}$, but $L_{TM} \not\leq_m \overline{L_{TM}}$

But, they have weaker implications than mapping reductions:

3. If $A \leq_T B$
   - If B is decidable then A is decidable
   - If A is not decidable, then B is not decidable

Turing Reduction Properties

Turing reductions are more general than mapping reductions:

1. If $A \leq_m B$, then $A \leq_T B$
2. If $A \leq_T B$, then it is not necessarily the case that $A \leq_m B$
    - In particular, $L_{TM} \leq_T \overline{L_{TM}}$, but $L_{TM} \not\leq_m \overline{L_{TM}}$

But, they have weaker implications than mapping reductions:

3. If $A \leq_T B$
    - If B is decidable then A is decidable
    - If A is not decidable, then B is not decidable
4. If $A \leq_T B$
    - If B is Turing-recognizable A is not necessarily Turing-recognizable

Turing Reduction Properties

Turing reductions are more general than mapping reductions:

1. If $A \leq_m B$, then $A \leq_T B$
2. If $A \leq_T B$, then it is not necessarily the case that $A \leq_m B$
   - In particular, $L_{TM} \leq_T \overline{L_{TM}}$, but $L_{TM} \not\leq_m \overline{L_{TM}}$

But, they have weaker implications than mapping reductions:

3. If $A \leq_T B$
   - If B is decidable then A is decidable
   - If A is not decidable, then B is not decidable
4. If $A \leq_T B$
   - If B is Turing-recognizable A is not necessarily Turing-recognizable
   - If A is not Turing-recognizable, cannot say if B is Turing-recognizable

Outline

Kolmogorov Complexity

Definition

Consider $x \in \{0,1\}^*$.

Kolmogorov Complexity

Definition

Consider $x \in \{0,1\}^*$.

1. The minimal description of x ($d(x)$) is the shortest string $\langle M, w \rangle$ such that TH M on input w halts with x on its tape

Kolmogorov Complexity

Definition

Consider $x \in \{0,1\}^*$.

1. The minimal description of x ($d(x)$) is the shortest string $\langle M, w \rangle$ such that TH M on input w halts with x on its tape
2. The Kolmogorov complexity of x is

$$K(x) = |d(x)|$$

Kolmogorov Complexity

Definition

Consider $x \in \{0,1\}^*$.

1. The minimal description of x ($d(x)$) is the shortest string $\langle M, w \rangle$ such that TH M on input w halts with x on its tape
2. The Kolmogorov complexity of x is

$$K(x) = |d(x)|$$

- $K(x)$ is the minimal description of x

Kolmogorov Complexity

Definition

Consider $x \in \{0,1\}^*$.

1. The minimal description of x ($d(x)$) is the shortest string $\langle M, w \rangle$ such that TH M on input w halts with x on its tape
2. The Kolmogorov complexity of x is

$$K(x) = |d(x)|$$

- $K(x)$ is the minimal description of x
- This captures the "amount of information" in x

What You Need to Know

- Basic definition of Kolmogorov complexity
- Be able to find rough bounds on Kolmogorov complexity
- Don't need to be able to prove anything