Computational Models — Lecture 8

Iftach Haitner.

Tel Aviv University.

December 11, 2017

1 Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.
Talk Outline

- Non-deterministic Turing machines
- Enumerators
- Decidability vs. Enumerability
- Encoding of Turing Machines and Universal Turing Machines
- The Halting/Acceptance problem
- Beyond Enumerable and co-Enumerable

- Sipser’s book, 3.2, 3.3, 4.1 and 4.2.
Part I

Non-deterministic Turing machines (NTMs)
Non-Deterministic Turing Machines (NTMs)

NTM $N = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$,

$$\delta : Q \times \Gamma \mapsto P(Q \times \Gamma \times \{L, R\})$$
Non-Deterministic Turing Machines (NTMs)

NTM \( N = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r) \),

\[ \delta : Q \times \Gamma \mapsto \mathcal{P}(Q \times \Gamma \times \{L, R\}) \]

The **yield relation** (for NTM):

The computation tree of \( N \) on input \( w \):

▶ Root is the starting configuration (with respect to \( w \))
▶ The children of a node are all configurations it (directly) yields.
# of children is at most: \(|Q| \cdot |\Gamma| \cdot 2.

Valid sequences of configurations with respect to \( N \) and \( w \), are defined as in the deterministic case.
Any rooted finite path in the computation tree of \( N \) and \( w \), it is a valid sequence of configurations (with respect to \( N \) and \( w \)).
Non-Deterministic Turing Machines (NTMs)

NTM $N = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$,

$$\delta : Q \times \Gamma \mapsto \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

The yield relation (for NTM): configuration $C$ yields $D$, if it yields it, according to the deterministic definition, for some deterministic restriction of $\delta$.

Configuration $C = (x'qx'')$ yields $D = (y'py'')$, if 
$$((p, (y'y'')_{\text{head}(C)}, X) \in \delta(q, (x'x'')_{\text{head}(C)}))$$ and \ldots
Non-Deterministic Turing Machines (NTMs)

NTM $N = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$,

$$\delta : Q \times \Gamma \mapsto \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

The yield relation (for NTM): configuration $C$ yields $D$, if it yields it, according to the deterministic definition, for some deterministic restriction of $\delta$.

Configuration $C = (x'qx'')$ yields $D = (y'py'')$, if $(p, (y'y'')_{\text{head}(C)}, X) \in \delta(q, (x'x'')_{\text{head}(C)})$ and . . .

The computation tree of $N$ on input $w$:

- **Root** is the starting configuration (with respect to $w$)
- **The children** of a node are all configurations it (directly) yields. 
  
  \# of children is at most: $|Q| \cdot |\Gamma| \cdot 2$. 

Non-Deterministic Turing Machines (NTMs)

NTM \( N = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r) \),

\[ \delta : Q \times \Gamma \mapsto \mathcal{P}(Q \times \Gamma \times \{L, R\}) \]

The yield relation (for NTM): configuration \( C \) yields \( D \), if it yields it, according to the deterministic definition, for some deterministic restriction of \( \delta \).

Configuration \( C = (x'qx'') \) yields \( D = (y'py'') \), if 
\( (p, (y'y'')_{\text{head}(C)}, X) \in \delta(q, (x'x'')_{\text{head}(C)}) \) and . . .

The computation tree of \( N \) on input \( w \):

- Root is the starting configuration (with respect to \( w \))
- The children of a node are all configurations it (directly) yields.
  - # of children is at most: \( |Q| \cdot |\Gamma| \cdot 2 \).

Valid sequences of configurations with respect to \( N \) and \( w \), are defined as in the deterministic case.
Non-Deterministic Turing Machines (NTMs)

NTM \( N = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r) \),

\[ \delta : Q \times \Gamma \mapsto \mathcal{P}(Q \times \Gamma \times \{L, R\}) \]

The yield relation (for NTM): configuration \( C \) yields \( D \), if it yields it, according to the deterministic definition, for some deterministic restriction of \( \delta \).

Configuration \( C = (x'qx''') \) yields \( D = (y'py''') \), if \( (p, (y'y'''))_{\text{head}(C), X} \in \delta(q, (x'x'''))_{\text{head}(C)} \) and . . .

The computation tree of \( N \) on input \( w \):

- **Root** is the starting configuration (with respect to \( w \))
- **The children** of a node are all configurations it (directly) yields.
  
  # of children is at most: \(|Q| \cdot |\Gamma| \cdot 2\).

Valid sequences of configurations with respect to \( N \) and \( w \), are defined as in the deterministic case.

Any rooted finite path in the computation tree of \( N \) and \( w \), it is a valid sequence of configurations (with respect to \( N \) and \( w \)).
Accepting a word

$N$ accepts $w \in \Sigma^*$, if $\exists$ an accepting path (i.e., accepting sequence of configurations) in its computation tree of $N$ on $w$.

(Equivalently, $\exists$ an accepting sequence of configurations, with respect to $N$ and $w$.)
Accepting a word

$N$ accepts $w \in \Sigma^*$, if $\exists$ an accepting path (i.e., accepting sequence of configurations) in its computation tree of $N$ on $w$.

(Equivalently, $\exists$ an accepting sequence of configurations, with respect to $N$ and $w$.)

$N$ halts on $w$, if it accepts it, or the computation tree of $N$ on $w$ is finite: $\exists k \in \mathbb{N}$, such that there is no valid sequence of length $k$. 

Accepting a word

$N$ accepts $w \in \Sigma^*$, if $\exists$ an accepting path (i.e., accepting sequence of configurations) in its computation tree of $N$ on $w$.

(Equivalently, $\exists$ an accepting sequence of configurations, with respect to $N$ and $w$.)

$N$ halts on $w$, if it accepts it, or the computation tree of $N$ on $w$ is finite: $\exists k \in \mathbb{N}$, such that there is no valid sequence of length $k$.

$N$ rejects $w$, if it halts but does not accept $w$. 
Equivalence of TM and NTM

It is clear that NTM is Turing complete.
Equivalence of TM and NTM

It is clear that NTM is Turing complete.

**Theorem 1**

For any non-deterministic TM there exists a deterministic TM that emulates it.
Equivalence of TM and NTM

It is clear that NTM is Turing complete.

**Theorem 1**

For any non-deterministic TM there exists a deterministic TM that emulates it.

**Corollary 2**

A language is enumerable [resp., decidable], if and only if there is some non-deterministic Turing machine that accepts [resp., decides] it.
Equivalence of **TM** and **NTM**

It is clear that NTM is Turing complete.

**Theorem 1**

For any non-deterministic **TM** there exists a deterministic **TM** that emulates it.

**Corollary 2**

A language is enumerable [resp., decidable], if and only if there is some non-deterministic Turing machine that accepts [resp., decides] it.

We will prove a slightly simpler to prove result.

**Theorem 3**

For any NTM $N$ there exists TM $D$ with $L(N) = L(D)$. 
Basic idea

- $D$ tries all possible branches in $N$ computation tree
- If $D$ finds any accepting path, it accepts.
Basic idea

- $D$ tries all possible branches in $N$ computation tree
- If $D$ finds any accepting path, it accepts.

Question 4

How to traverse this tree?
- depth-first search?
- breadth-first search?
The machine $D$ has three tapes:

- **Input** tape is never altered (only read from),
- **Emulation** tape serves as $N$’s tape,
- **Address** tape keeps track of $D$’s location in $N$’s *computation tree.*
Address tape

Let \( b \) be bound on the \# of children of node in \( N \)'s computation tree.
Address tape

Let $b$ be bound on the number of children of node in $N$'s computation tree. The address tape contains a pointer into the configuration tree.
Address tape

Let $b$ be bound on the number of children of node in $N$'s computation tree. The address tape contains a pointer into the configuration tree. Concretely, a string in $\Sigma^*_b$, for $\Sigma_b = \{1, \ldots, b\}$.

**Definition 5**

By incrementing the value of the address tape, we mean replace its content with the next string in $\Sigma^*_b$, according to the lexicographic order.

**Example** (for $b = 2$):

- $\varepsilon \mapsto 1$
- $1 \mapsto 2$
- $2 \mapsto 11$
- $11 \mapsto 12$
- $12 \mapsto 21$
- $21 \mapsto 22$
- $22 \mapsto 111$ . . .

**Question 6**

Can a TM implement the increment function?

**Question 7**

Can a (deterministic) TM compute the value of the node indexed by the address tape (with respect to TM $N$ and input $w$)?
Address tape

Let $b$ be bound on the number of children of node in $N$'s computation tree.

The address tape contains a pointer into the configuration tree. Concretely, a string in $\Sigma^*$, for $\Sigma_b = \{1, \ldots, b\}$.

**Definition 5**

By incrementing the value of the address tape, we mean replace its content with the next string in $\Sigma_b^*$, according to the lexicographic order.
Address tape

Let $b$ be bound on the number of children of node in $N$’s computation tree.

The address tape contains a pointer into the configuration tree. Concretely, a string in $\Sigma_b^*$, for $\Sigma_b = \{1, \ldots, b\}$.

**Definition 5**

By **incrementing** the value of the address tape, we mean replace its content with the next string in $\Sigma_b^*$, according to the lexicographic order.

Example (for $b = 2$): $\varepsilon \mapsto 1 \mapsto 2 \mapsto 11 \mapsto 12 \mapsto 21 \mapsto 22 \mapsto 111 \ldots$

**Question 6**

Can a TM implement the increment function?

**Question 7**

Can a (deterministic) TM compute the value of the node indexed by the address tape (with respect to TM $N$ and input $w$)?
Address tape

Let $b$ be bound on the number of children of node in $N$’s computation tree.

The address tape contains a pointer into the configuration tree. Concretely, a string in $\Sigma_b^*$, for $\Sigma_b = \{1, \ldots, b\}$.

**Definition 5**

By incrementing the value of the address tape, we mean replace its content with the next string in $\Sigma_b^*$, according to the lexicographic order.

Example (for $b = 2$): $\varepsilon \mapsto 1 \mapsto 2 \mapsto 11 \mapsto 12 \mapsto 21 \mapsto 22 \mapsto 111 \ldots$

**Question 6**

Can a TM implement the increment function?

**Question 7**

Can a (deterministic) TM compute the value of the node indexed by the address tape (with respect to TM $N$ and input $w$)?
Address tape

Let $b$ be bound on the # of children of node in $N$’s computation tree.

The address tape contains a pointer into the configuration tree. Concretely, a string in $\Sigma_b^*$, for $\Sigma_b = \{1, \ldots, b\}$.

Definition 5

By incrementing the value of the address tape, we mean replace its content with the next string in $\Sigma_b^*$, according to the lexicographic order.

Example (for $b = 2$): $\varepsilon \rightarrow 1 \rightarrow 2 \rightarrow 11 \rightarrow 12 \rightarrow 21 \rightarrow 22 \rightarrow 111 \ldots$

Question 6

Can a TM implement the increment function?

Question 7

Can a (deterministic) TM compute the value of the node indexed by the address tape (with respect to TM $N$ and input $w$)?
Algorithm 8 (TM $D$ (pseudocode))

1. Compute the configuration of $N$ indexed by the $address$ tape:
   1.1 Copy input tape (i.e., $w$) to emulation tape.
   1.2 Use emulation tape to emulate the run of $N$ on $w$, using the address tape to resolve non-deterministic choices.
      Break current emulation, if
      ★ End of path (i.e., symbols on address tape are exhausted)
      ★ Non-deterministic choice is invalid

2. Accept, if an accepting configuration was reached.

3. Increment the value of address tape.

4. Go back to Step 1.
Algorithm 8 (TM $D$ (pseudocode))

1. Compute the configuration of $N$ indexed by the $address$ tape:
   1.1 Copy input tape (i.e., $w$) to emulation tape.
   1.2 Use emulation tape to emulate the run of $N$ on $w$, using the $address$ tape to resolve non-deterministic choices.

   Break current emulation, if
   - End of path (i.e., symbols on address tape are exhausted)
   - Non-deterministic choice is invalid

2. Accept, if an accepting configuration was reached.

3. Increment the value of $address$ tape.

4. Go back to Step 1.

Question 9

Change $D$ to emulate $N$. 
Part II

Enumerators
Enumerators

A language is enumerable, if it is accepted by some Turing Machine.
Enumerators

A language is **enumerable**, if it is accepted by some Turing Machine.

**Question 10**

But why enumerable?
Enumerators

A language is **enumerable**, if it is accepted by some Turing Machine.

**Question 10**

But why enumerable?

A TM is an **enumerator** for a language $L$, if on the empty input strings, it outputs **all** the strings in $L$ and **nothing else**.
Enumerators

A language is enumerable, if it is accepted by some Turing Machine.

Question 10

But why enumerable?

A TM is an enumerator for a language $L$, if on the empty input strings, it outputs all the strings in $L$ and nothing else.
**Definition 11 (enumerator)**

A deterministic TM $M$ is an *enumerator* for a language $L \subseteq \Sigma^*$, if on the empty input string it does as follows:

- On its first tape (i.e., output tape), $M$
  - Writes only elements from $\Sigma \cup \$$(we assume wlg. $\$ \notin \Sigma$)
  - Never alters cell more than once (i.e., write only tape)
- Every word in $L$ appears after a finite number of steps on the output tap (i.e., between two $\$’s)
- Word not in $L$ never appears on the output tape
Theorem 12

A language is in $\mathcal{RE}$ iff it has an enumerator.
Having Enumerator ⇔ Being in \( \mathcal{RE} \)

**Theorem 12**

A language is in \( \mathcal{RE} \) iff it has an enumerator.

Will show

- If \( E \) enumerates language \( L \), then some TM \( M \) accepts \( L \).
- If \( M \) accepts \( L \), then some enumerator \( E \) enumerates it.
Claim 13

If a TM $E$ enumerates a language $L$, then some TM $M$ accepts $L$. 

Proof:
Algorithm 14 (TM $M$)
On input $w$, run $E$.
Every time $E$ outputs a string $v$:
If $v = w$, accept.
Otherwise continue ♣
Claim 13

If a TM $E$ enumerates a language $L$, then some TM $M$ accepts $L$.

Proof:

**Algorithm 14 (TM $M$)**

On input $w$, run $E$.

Every time $E$ outputs a string $v$:

- If $v = w$, accept.
- Otherwise continue.
Claim 15

If a TM $M$ accepts $L$, then some enumerator TM $E$ enumerates $L$. 

Proof: Let $s_1, s_2, s_3, ...$ be a list of all strings in $\Sigma^*$ (e.g., strings in lexicographic order).

Algorithm 16 (TM $E$)

Repeat the following for $i = 1, 2, 3, ...$

- Run $M$ for $i$ steps on each input $s_1, s_2, ..., s_i$.
- For any accepting computation, output the corresponding $s$.

Note that with this procedure, each output is duplicated infinitely often.

Question 17 Can this duplication be avoided?

Question 18 Can we do the enumeration in (lexicographic) order?
Being in $\mathcal{RE} \iff$ Having Enumerator

Claim 15

If a TM $M$ accepts $L$, then some enumerator TM $E$ enumerates $L$.

Proof: Let $s_1, s_2, s_3, \ldots$ be a list of all strings in $\Sigma^*$ (e.g., strings in lexicographic order).
Claim 15

If a TM $M$ accepts $L$, then some enumerator TM $E$ enumerates $L$.

Proof: Let $s_1, s_2, s_3, \ldots$ be a list of all strings in $\Sigma^*$ (e.g., strings in lexicographic order).

Algorithm 16 (TM $E$)

Repeat the following for $i = 1, 2, 3, \ldots$

- Run $M$ for $i$ steps on each input $s_1, s_2, \ldots, s_i$.

  For any accepting computation, output the corresponding $s$.

♣

Note that with this procedure, each output is duplicated infinitely often.

Question 17 Can this duplication be avoided?

Question 18 Can we do the enumeration in (lexicographic) order?
Claim 15
If a TM $M$ accepts $L$, then some enumerator TM $E$ enumerates $L$.

Proof: Let $s_1, s_2, s_3, \ldots$ be a list of all strings in $\Sigma^*$ (e.g., strings in lexicographic order).

Algorithm 16 (TM $E$)
Repeat the following for $i = 1, 2, 3, \ldots$

- Run $M$ for $i$ steps on each input $s_1, s_2, \ldots, s_i$.
  For any accepting computation, output the corresponding $s$.

Note that with this procedure, each output is duplicated infinitely often.
Claim 15

If a TM \( M \) accepts \( L \), then some enumerator TM \( E \) enumerates \( L \).

Proof: Let \( s_1, s_2, s_3, \ldots \) be a list of all strings in \( \Sigma^* \) (e.g., strings in lexicographic order).

Algorithm 16 (TM \( E \))

Repeat the following for \( i = 1, 2, 3, \ldots \)

- Run \( M \) for \( i \) steps on each input \( s_1, s_2, \ldots, s_i \).

For any accepting computation, output the corresponding \( s \).

Note that with this procedure, each output is duplicated infinitely often.

Question 17

Can this duplication be avoided?
Being in \( \mathcal{RE} \) \( \quad \rightarrow \quad \) Having Enumerator

**Claim 15**

If a TM \( M \) accepts \( L \), then some enumerator TM \( E \) enumerates \( L \).

**Proof:** Let \( s_1, s_2, s_3, \ldots \) be a list of all strings in \( \Sigma^* \) (e.g., strings in lexicographic order).

**Algorithm 16 (TM \( E \))**

Repeat the following for \( i = 1, 2, 3, \ldots \)

- Run \( M \) for \( i \) steps on each input \( s_1, s_2, \ldots, s_i \).

  For any accepting computation, output the corresponding \( s \).

\[\clubsuit\]

Note that with this procedure, each output is duplicated infinitely often.

**Question 17**

Can this duplication be avoided?

**Question 18**

Can we do the enumeration in (lexicographic) order?
Having “In order” Enumerator ⇔ Being in \( \mathcal{R} \)

**Theorem 19**

A language \( L \) is **decidable** iff \( \exists \) an enumerator that enumerates \( L \) in **lexicographic order**.
A language $L$ is decidable iff $\exists$ an enumerator that enumerates $L$ in lexicographic order.

Proof: ?
Having “In order” Enumerator $\iff$ Being in $\mathcal{R}$

**Theorem 19**

A language $L$ is **decidable** iff $\exists$ an enumerator that enumerates $L$ in **lexicographic order**.

**Proof**: Left as an exercise. ♣
Part III

Decidability vs. Enumerability
Decidability vs. Enumerability

- $\mathcal{RE}$ – the class of enumerable languages
Decidability vs. Enumerability

- $\mathcal{RE}$ – the class of enumerable languages
- $\text{co-RE} = \{L: \overline{L} \in \mathcal{RE}\}$ — the class of languages whose complement is enumerable.
Decidability vs. Enumerability

- $\mathcal{RE}$ – the class of enumerable languages
- $\text{co-RE} = \{L : \overline{L} \in \mathcal{RE}\}$ — the class of languages whose complement is enumerable.
- $\mathcal{R}$ – the class of decidable languages.

Claim 20: $\mathcal{R} \subseteq \mathcal{RE} \cap \text{co-RE}$.

Proof:
- If $L \in \mathcal{R}$, then $L \in \mathcal{RE}$.
- If $L \in \mathcal{R}$, then $L \in \mathcal{RE} \subseteq \text{co-RE}$.
Decidability vs. Enumerability

- $\mathcal{RE}$ – the class of enumerable languages
- $co-\mathcal{RE} = \{L: \overline{L} \in \mathcal{RE}\}$ — the class of languages whose complement is enumerable.
- $\mathcal{R}$ – the class of decidable languages.

Claim 20: $\mathcal{R} \subseteq \mathcal{RE} \cap co-\mathcal{RE}$.

Proof:

1. $L \in \mathcal{R} \Rightarrow L \in \mathcal{RE}$
2. $L \in \mathcal{R} \Rightarrow \overline{L} \in \mathcal{RE}$
3. $\overline{L} \in \mathcal{RE} \Rightarrow L \in co-\mathcal{RE}$
Decidability vs. Enumerability

- $\mathcal{RE}$ – the class of enumerable languages
- $\text{co-RE} = \{L: \overline{L} \in \mathcal{RE}\}$ — the class of languages whose complement is enumerable.
- $\mathcal{R}$ – the class of decidable languages.

Claim 20

$\mathcal{R} \subseteq \mathcal{RE} \cap \text{co-RE}$. 
Decidability vs. Enumerability

- $\mathcal{RE}$ – the class of enumerable languages
- $\text{co-RE} = \{ L : \overline{L} \in \mathcal{RE} \}$ — the class of languages whose complement is enumerable.
- $\mathcal{R}$ – the class of decidable languages.

Claim 20

$$\mathcal{R} \subseteq \mathcal{RE} \cap \text{co-RE}.$$  

Proof:

- $L \in \mathcal{R} \implies L \in \mathcal{RE}$
Decidability vs. Enumerability

- $\mathcal{RE}$ – the class of enumerable languages
- $\text{co-RE} = \{L : \overline{L} \in \mathcal{RE}\}$ — the class of languages whose complement is enumerable.
- $\mathcal{R}$ – the class of decidable languages.

Claim 20

$\mathcal{R} \subseteq \mathcal{RE} \cap \text{co-RE}$. 

Proof:

- $L \in \mathcal{R} \implies L \in \mathcal{RE}$
- $L \in \mathcal{R} \implies \overline{L} \in \mathcal{R}$
Decidability vs. Enumerability

- \( \mathcal{RE} \) – the class of enumerable languages
- \( \text{co-RE} = \{ L : \overline{L} \in \mathcal{RE} \} \) — the class of languages whose complement is enumerable.
- \( \mathcal{R} \) – the class of decidable languages.

**Claim 20**

\( \mathcal{R} \subseteq \mathcal{RE} \cap \text{co-RE} \).

Proof:

- \( L \in \mathcal{R} \implies L \in \mathcal{RE} \)
- \( L \in \mathcal{R} \implies \overline{L} \in \mathcal{R} \)
Decidability vs. Enumerability

- $\mathcal{RE}$ – the class of enumerable languages
- $\text{co-RE} = \{L : \overline{L} \in \mathcal{RE}\}$ – the class of languages whose complement is enumerable.
- $\mathcal{R}$ – the class of decidable languages.

**Claim 20**

$\mathcal{R} \subseteq \mathcal{RE} \cap \text{co-RE}$. 

**Proof:**

- $L \in \mathcal{R} \implies L \in \mathcal{RE}$
- $L \in \mathcal{R} \implies \overline{L} \in \mathcal{R} \implies \overline{L} \in \mathcal{RE} \implies L \in \text{co-RE}$
Theorem \( \mathcal{R} = \mathcal{R}E \cap \text{co-RE} \)

Claim 21

Combing with previous claims, it follows that

Theorem 22

\( \mathcal{R} = \mathcal{R}E \cap \text{co-RE} \).

Proof:

For \( L \in \mathcal{R}E \cap \text{co-RE} \), let \( M_1 \) be a TM that accepts \( L \), and let \( M_2 \) be a TM that accepts \( L \).

Algorithm 23 (\( M \)-a decider for \( L \))

Input: \( w \).

\[ \begin{align*}
\text{Run both } M_1 \text{ and } M_2 \text{ in "in parallel".} \\
\text{Accept if } M_1 \text{ accepts} \\
\text{Reject if } M_2 \text{ accepts.}
\end{align*} \]
Theorem $\mathcal{R} = \mathcal{RE} \cap \text{co-RE}$

Claim 21

$\mathcal{R} \supseteq \mathcal{RE} \cap \text{co-RE}.$

Combining with previous claims, it follows that

Theorem 22

$\mathcal{R} = \mathcal{RE} \cap \text{co-RE}.$
Theorem $\mathcal{R} = \text{RE} \cap \text{co-RE}$

Claim 21

$\mathcal{R} \supseteq \text{RE} \cap \text{co-RE}$.

Combing with previous claims, it follows that

Theorem 22

$\mathcal{R} = \text{RE} \cap \text{co-RE}$.

Proof: (of claim)
**Theorem** $\mathcal{R} = \mathcal{RE} \cap \text{co-}\mathcal{RE}$

**Claim 21**

$\mathcal{R} \supseteq \mathcal{RE} \cap \text{co-}\mathcal{RE}$.

Combing with previous claims, it follows that

**Theorem 22**

$\mathcal{R} = \mathcal{RE} \cap \text{co-}\mathcal{RE}$.

**Proof: (of claim)**

For $L \in \mathcal{RE} \cap \text{co-}\mathcal{RE}$, let $M_1$ be a TM that accepts $L$, and let $M_2$ be a TM that accepts $\overline{L}$.
Theorem \( R = \text{RE} \cap \text{co-RE} \)

Claim 21
\( R \supseteq \text{RE} \cap \text{co-RE} \).

Combing with previous claims, it follows that

Theorem 22
\( R = \text{RE} \cap \text{co-RE} \).

Proof: (of claim)
For \( L \in \text{RE} \cap \text{co-RE} \), let \( M_1 \) be a TM that accepts \( L \), and let \( M_2 \) be a TM that accepts \( \overline{L} \).

Algorithm 23 (\( M \) - a decider for \( L \))
Input: \( w \).
   - Run both \( M_1 \) and \( M_2 \) in “in parallel”.
   - Accept if \( M_1 \) accepts
   - Reject if \( M_2 \) accepts
Claim 24

$M$ decides $L$

Proof:

Every string is in either $L$ or $L'$ (of course not in both).

Thus either $M_1$ or $M_2$ accepts the input $w$.

Since $M$ halts whenever $M_1$ or $M_2$ accepts, $M$ always halts (and hence is a decider).

Moreover, $M$ accepts strings in $L$ and rejects strings in $L'$. 
$M$ decides $L$

Claim 24

$M$ decides $L$

Proof:

- Every string is in either $L$ or in $\overline{L}$ (of course not in both).

♣
$M$ decides $L$

Claim 24

$M$ decides $L$

Proof:

- Every string is in either $L$ or in $\overline{L}$ (of course not in both).
- Thus either $M_1$ or $M_2$ accepts the input $w$. 
Claim 24

\( M \) decides \( L \)

Proof:

- Every string is in either \( L \) or in \( \overline{L} \) (of course not in both).
- Thus either \( M_1 \) or \( M_2 \) accepts the input \( w \).
- Since \( M \) halts whenever \( M_1 \) or \( M_2 \) accepts, \( M \) always halts (and hence is a decider).
$M$ decides $L$

**Claim 24**

$M$ decides $L$

**Proof:**
- Every string is in either $L$ or in $\overline{L}$ (of course not in both).
- Thus either $M_1$ or $M_2$ accepts the input $w$.
- Since $M$ halts whenever $M_1$ or $M_2$ accepts, $M$ always halts (and hence is a decider).
- Moreover, $M$ accepts strings in $L$ and rejects strings in $\overline{L}$.

♣
Emulating TM’s in parallel

Question 25
What does it mean to emulate $M_1, M_2$ in parallel?

Algorithm 26 (TM $M$)

Do (forever)

1. Emulate the next step of $M_1$
2. Emulate the next step of $M_2$
3. If this is accepting configuration for some $M_i$, halt and return $i$. 

Iftach Haitner (TAU)
Computational Models, Lecture 8
December 11, 2017 22 / 52
Emulating TM’s in parallel

Question 25

What does it mean to emulate $M_1, M_2$ in parallel?

Answer: $M$ has two tapes, one for each machine.

Algorithm 26 (TM $M$)

Do (forever)

1. Emulate the next step of $M_1$
2. Emulate the next step of $M_2$
3. If this is accepting configuration for some $M_i$, halt and return $i$. 
How do we know that CFL (and thus regular languages) are in $\mathcal{R}$?
Question 27

How do we know that CFL (and thus regular languages) are in \( \mathcal{R} \)?
Part IV

Encodings and Universal TM
Encodings

- Input to a Turing machine is a string of symbols.
Encodings

- Input to a Turing machine is a \textit{string of symbols}.
- We want algorithms that work on graphs, matrices, polynomials, Turing machines, etc.

Need to choose an encoding for objects (can often be done in many reasonable ways).

Sometimes it is helpful to distinguish between $X$, the object, and $\langle X \rangle$, its encoding.
Encodings

- Input to a Turing machine is a string of symbols.
- We want algorithms that work on graphs, matrices, polynomials, Turing machines, etc.
- Need to choose an encoding for objects (can often be done in many reasonable ways).
Encodings

- Input to a Turing machine is a string of symbols.
- We want algorithms that work on graphs, matrices, polynomials, Turing machines, etc.
- Need to choose an encoding for objects (can often be done in many reasonable ways).
- Sometimes it is helpful to distinguish between $X$, the object, and $\langle X \rangle$, its encoding.
Encoding of Turing Machines

Turing machines can be encoded as strings.
Encoding of Turing Machines

Turing machines can be encoded as strings.

Such encoding will enable us

- To check (by an algorithm) that a given string is a legal encoding of a TM. *(Similar to a compiler checking for syntax errors.)*
- To build a universal machine that can read such encoding and emulates the encoded TM on any input string. *(Similar to running an interpreter.)*
### Definition 28 (Encoding of \( \langle M \rangle \) of a TM \( M \))

Let \( M = (Q, \Sigma = \{\sigma_1, \ldots, \sigma_k\}, \Gamma, \delta, q_0, q_a, q_r) \) be a TM. Assume wlg. that

1. \( Q = \{q_1, \ldots, q_m\} \), where \( q_0, q_a \) and \( q_r \) are indicated by \( q_1, q_2 \) and \( q_3 \).
2. \( \Gamma = \{\gamma_1, \ldots, \gamma_s\} \), where \( \sigma_1, \ldots, \sigma_k, \omega \) are indicated by \( \gamma_1, \ldots, \gamma_{k+1} \).
3. The directions \( L \) and \( R \) be indicated by \( D_1 \) and \( D_2 \).
Definition 28 (Encoding of $\langle M \rangle$ of a TM $M$)

Let $M = (Q, \Sigma = \{\sigma_1, \ldots, \sigma_k\}, \Gamma, \delta, q_0, q_a, q_r)$ be a TM. Assume w.l.o.g. that

- $Q = \{q_1, \ldots, q_m\}$, where $q_0$, $q_a$ and $q_r$ are indicated by $q_1$, $q_2$ and $q_3$.
- $\Gamma = \{\gamma_1, \ldots, \gamma_s\}$, where $\sigma_1, \ldots, \sigma_k$, $\$\$ are indicated by $\gamma_1, \ldots, \gamma_{k+1}$.
- The directions $L$ and $R$ be indicated by $D_1$ and $D_2$.

To encode $M$, we only encode the transition function $\delta$. For each rule $\delta(q_i, \gamma_j) = (q_k, \gamma_\ell, D_b)$, we add the string $0^i 10^j 10^k 10^\ell 10^b$. 

Iftach Haitner (TAU)
# Standard Encoding of Turing Machines

## Definition 28 (Encoding of $\langle M \rangle$ of a TM $M$)

Let $M = (Q, \Sigma = \{\sigma_1, \ldots, \sigma_k\}, \Gamma, \delta, q_0, q_a, q_r)$ be a TM. Assume wlg. that

- $Q = \{q_1, \ldots, q_m\}$, where $q_0, q_a$ and $q_r$ are indicated by $q_1, q_2$ and $q_3$.
- $\Gamma = \{\gamma_1, \ldots, \gamma_s\}$, where $\sigma_1, \ldots, \sigma_k, \sqcup$ are indicated by $\gamma_1, \ldots, \gamma_{k+1}$.
- The directions $L$ and $R$ be indicated by $D_1$ and $D_2$.

To encode $M$, we only encode the transition function $\delta$. For each rule $\delta(q_i, \gamma_j) = (q_k, \gamma_\ell, D_b)$, we add the string $0^i 10^j 10^k 10^\ell 10^b$.

Different rules are separated by 11.
Standard Encoding of Turing Machines

**Definition 28 (Encoding of \( \langle M \rangle \) of a TM \( M \))**

Let \( M = (Q, \Sigma = \{\sigma_1, \ldots, \sigma_k\}, \Gamma, \delta, q_0, q_a, q_r) \) be a TM. Assume w.l.g. that

- \( Q = \{q_1, \ldots, q_m\} \), where \( q_0, q_a \) and \( q_r \) are indicated by \( q_1, q_2 \) and \( q_3 \).
- \( \Gamma = \{\gamma_1, \ldots, \gamma_s\} \), where \( \sigma_1, \ldots, \sigma_k, \sqcup \) are indicated by \( \gamma_1, \ldots, \gamma_{k+1} \).
- The directions \( L \) and \( R \) be indicated by \( D_1 \) and \( D_2 \).

To encode \( M \), we only encode the transition function \( \delta \). For each rule \( \delta(q_i, \gamma_j) = (q_k, \gamma_\ell, D_b) \), we add the string \( 0^i 10^j 10^k 10^\ell 10^b \).

Different rules are separated by 11.

**Fact 29**

*There exists a TM (called universal TM) that on input \( \langle M, w \rangle \) (encoded by \( (1100*100*100*100*100*)*111(0 \cup 1)^* \), can check that \( \langle M \rangle \) encoded a TM, and can emulates \( M(w) \).*
The Universal Turing Machine
Algorithm 30 (Universal TM $U$)

On input $\langle M, w \rangle$, where $\langle M \rangle$ and $\langle w \rangle$ are binary strings separated by 111.

- Checks that $\langle M, w \rangle$ is a proper encoding of a TM.
- Emulate $M(w)$ (how?)
  - Accept, if $M$ enters its accept state
  - Reject, if $M$ enters its reject state
Universal Turing Machines

Algorithm 30 (Universal TM $U$)

On input $\langle M, w \rangle$, where $\langle M \rangle$ and $\langle w \rangle$ are binary strings separated by 111.

- Checks that $\langle M, w \rangle$ is a proper encoding of a TM.
- Emulate $M(w)$ (how?)
  - Accept, if $M$ enters its accept state
  - Reject, if $M$ enters its reject state

Notice that as a consequence, if $M$ on input $w$ enters an infinite loop, so does $U$ on input $\langle M, w \rangle$. 
Universal Turing Machines (2)

- The universal machine $U$ obviously has a **fixed number** of states (100 should do).
- Despite this, it can simulate machines $M$ with many more states.
- Universal machines inspired the development of stored-program computers in the 40s and 50s.
- Most of you have **seen** a universal machine, and have even **used** one!
Universal Turing Machines (3)

- For example, *Dr. Scheme* (interpreter) is a universal *Scheme* machine.

- It accepts a two part input: “Above the line” – the program (corresponding to $\langle M \rangle$), and “below the line” the input to run it on (corresponding to $w$).
Part V

The Acceptance & Halting Problems
The Acceptance & Halting Problems

- Of the most philosophically important theorems of the theory of computation.
The Acceptance & Halting Problems

- Of the most philosophically important theorems of the theory of computation.
- Computers (and computation) are not omnipotent – they are limited in a very fundamental way.
The Acceptance & Halting Problems

- Of the most philosophically important theorems of the theory of computation.
- Computers (and computation) are not omnipotent – they are limited in a very fundamental way.
- Many common problems are unsolvable, e.g., does a program sort an array of integers?
The Acceptance & Halting Problems

- Of the most philosophically important theorems of the theory of computation.

- Computers (and computation) are not omnipotent – they are limited in a very fundamental way.

- Many common problems are unsolvable, e.g., does a program sort an array of integers?

  Note that these problems are well defined: both program and specification are precise mathematical objects.
The Acceptance & Halting Problems

- Of the most philosophically important theorems of the theory of computation.
- Computers (and computation) are not omnipotent – they are limited in a very fundamental way.
- Many common problems are unsolvable, e.g., does a program sort an array of integers?
  
  Note that these problems are well defined: both program and specification are precise mathematical objects.

Hey, proving program $\cong$ specification should be just like proving that triangle $1 \cong$ triangle $2 \ldots$
The Acceptance & Halting Problems

▶ Of the most philosophically important theorems of the theory of computation.

▶ Computers (and computation) are not omnipotent – they are limited in a very fundamental way.

▶ Many common problems are unsolvable, e.g., does a program sort an array of integers?

\[ \text{Note that these problems are well defined: both program and specification are precise mathematical objects.} \]

Hey, proving program \( \cong \) specification should be just like proving that triangle \( 1 \cong \) triangle \( 2 \ldots \)

Well, this is not the case!
CFG, NFA, DFA Reminders

We saw that the following language are decidable:

▶ A DFA = \{⟨M, w⟩: M is a DFA accepting the string w}\}.

▶ A NFA = \{⟨M, w⟩: M is an NFA accepting the string w}\}.

▶ A CFG = \{⟨M, w⟩: M is a PDA accepting the string w}\}.

▶ EMPTY CFG = \{⟨G⟩: G is a CFG \& L(G) = \emptyset\}.

What would happen with Turing Machines?

A TM = \{⟨M, w⟩: M is a TM that accepts w\}.

Theorem 31 (The Acceptance Problem is undecidable)

A TM is undecidable (i.e., no TM decides it).
We saw that the following language are decidable:

- $A_{\text{DFA}} = \{\langle M, w \rangle : M \text{ is a DFA accepting the string } w \}$. 
- $A_{\text{NFA}} = \{\langle M, w \rangle : M \text{ is an NFA accepting the string } w \}$. 
- $A_{\text{CFG}} = \{\langle M, w \rangle : M \text{ is a PDA accepting the string } w \}$. 
- $\text{EMPTY}_{\text{CFG}} = \{\langle G \rangle : G \text{ is a CFG} \land L(G) = \emptyset \}$. 

What would happen with Turing Machines?

- $A_{\text{TM}} = \{\langle M, w \rangle : M \text{ is a TM that accepts } w \}$. 

Theorem 31 (The Acceptance Problem is undecidable)

- $A_{\text{TM}}$ is undecidable (i.e., no TM decides it).
CFG, NFA, DFA Reminders

We saw that the following language are decidable:

- $A_{\text{DFA}} = \{ \langle M, w \rangle : M \text{ is a DFA accepting the string } w \}$. 
- $A_{\text{NFA}} = \{ \langle M, w \rangle : M \text{ is an NFA accepting the string } w \}$. 

What would happen with Turing Machines?

$A_{\text{TM}} = \{ \langle M, w \rangle : M \text{ is a TM that accepts } w \}$

Theorem 31 (The Acceptance Problem is undecidable)

$A_{\text{TM}}$ is undecidable (i.e., no TM decides it).
CFG, NFA, DFA Reminders

We saw that the following language are decidable:

- \( \mathcal{A}_{\text{DFA}} = \{ \langle M, w \rangle : M \text{ is a DFA accepting the string } w \} \).
- \( \mathcal{A}_{\text{NFA}} = \{ \langle M, w \rangle : M \text{ is an NFA accepting the string } w \} \).
- \( \mathcal{A}_{\text{CFG}} = \{ \langle M, w \rangle : M \text{ is a PDA accepting the string } w \} \).
CFG, NFA, DFA Reminders

We saw that the following language are decidable:

- \( A_{\text{DFA}} = \{ \langle M, w \rangle : M \text{ is a DFA accepting the string } w \} \).
- \( A_{\text{NFA}} = \{ \langle M, w \rangle : M \text{ is an NFA accepting the string } w \} \).
- \( A_{\text{CFG}} = \{ \langle M, w \rangle : M \text{ is a PDA accepting the string } w \} \).
- \( \text{EMPTY}_{\text{CFG}} = \{ \langle G \rangle : G \text{ is a CFG } \land L(G) = \emptyset \} \).
We saw that the following languages are decidable:

- $A_{\text{DFA}} = \{ \langle M, w \rangle : M \text{ is a DFA accepting the string } w \}.$
- $A_{\text{NFA}} = \{ \langle M, w \rangle : M \text{ is an NFA accepting the string } w \}.$
- $A_{\text{CFG}} = \{ \langle M, w \rangle : M \text{ is a PDA accepting the string } w \}.$
- $\text{EMPTY}_{\text{CFG}} = \{ \langle G \rangle : G \text{ is a CFG } \land L(G) = \emptyset \}.$

What would happen with Turing Machines?

- $A_{\text{TM}} = \{ \langle M, w \rangle : M \text{ is a TM that accepts } w \}.$

Theorem 31 (The Acceptance Problem is undecidable)

$A_{\text{TM}}$ is undecidable (i.e., no TM decides it).
We saw that the following language are **decidable**:

- \( A_{\text{DFA}} = \{ \langle M, w \rangle : M \text{ is a DFA accepting the string } w \} \).
- \( A_{\text{NFA}} = \{ \langle M, w \rangle : M \text{ is an NFA accepting the string } w \} \).
- \( A_{\text{CFG}} = \{ \langle M, w \rangle : M \text{ is a PDA accepting the string } w \} \).
- \( \text{EMPTY}_{\text{CFG}} = \{ \langle G \rangle : G \text{ is a CFG } \land L(G) = \emptyset \} \).

What would happen with Turing Machines?
CFG, NFA, DFA Reminders

We saw that the following language are decidable:

- $A_{DFA} = \{\langle M, w \rangle : M \text{ is a DFA accepting the string } w \}$.
- $A_{NFA} = \{\langle M, w \rangle : M \text{ is an NFA accepting the string } w \}$.
- $A_{CFG} = \{\langle M, w \rangle : M \text{ is a PDA accepting the string } w \}$.
- $\text{EMPTY}_{CFG} = \{\langle G \rangle : G \text{ is a CFG} \land L(G) = \emptyset \}$.

What would happen with Turing Machines?

- $A_{TM} = \{\langle M, w \rangle : M \text{ is a TM that accepts } w \}$.
CFG, NFA, DFA Reminders

We saw that the following language are decidable:

- \( A_{\text{DFA}} = \{ \langle M, w \rangle : M \text{ is a DFA accepting the string } w \} \).
- \( A_{\text{NFA}} = \{ \langle M, w \rangle : M \text{ is an NFA accepting the string } w \} \).
- \( A_{\text{CFG}} = \{ \langle M, w \rangle : M \text{ is a PDA accepting the string } w \} \).
- \( \text{EMPTY}_{\text{CFG}} = \{ \langle G \rangle : G \text{ is a CFG } \land \text{L}(G) = \emptyset \} \).

What would happen with Turing Machines?

\[ A_{\text{TM}} = \{ \langle M, w \rangle : M \text{ is a TM that accepts } w \} \]

Theorem 31 (The Acceptance Problem is undecidable)

\( A_{\text{TM}} \) is undecidable (i.e., no TM decides it).
The Acceptance problem

\[ \text{A}_{\text{TM}} = \{ \langle M, w \rangle : M \text{ is a TM that accepts } w \} \]

Before approaching the proof of undecidability, we first prove

**Theorem 32**

\( \text{A}_{\text{TM}} \) is recursively enumerable (namely in \( \mathcal{RE} \)).
The Acceptance problem

\[ A_{TM} = \{ \langle M, w \rangle : M \text{ is a TM that accepts } w \} \]

Before approaching the proof of undecidability, we first prove

**Theorem 32**

\( A_{TM} \) is recursively enumerable (namely in \( R.E \)).

**Proof:** The universal machine accepts \( A_{TM} \). ♣
Proving Thm 31

Suppose a TM, $H$, is a decider for $A_{TM}$. Namely,

$$H(\langle M, w \rangle) = \begin{cases} \text{accepts} & \text{if } M \text{ accepts } w \\ \text{rejects} & \text{if } M \text{ does not accept } w \end{cases}$$
Proving Thm 31

Suppose a TM, $H$, is a decider for $A_{TM}$. Namely,

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

Algorithm 33 ($D$)

On input $\langle M \rangle$

- Run $H$ on input $\langle M, M \rangle$.
- Output the opposite of what $H$ outputs:
  - Reject if $H$ accepts, and
  - Accept if $H$ rejects.
Proving Thm 31

Suppose a TM, $H$, is a decider for $A_{TM}$. Namely,

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

**Algorithm 33 ($D$)**

On input $\langle M \rangle$

- Run $H$ on input $\langle M, M \rangle$.
- Output the opposite of what $H$ outputs:
  - Reject if $H$ accepts, and
  - Accept if $H$ rejects.

What happens if we run $D$ on its own description?
Proving Thm 31

Suppose a TM, $H$, is a decider for $A_{TM}$. Namely,

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

Algorithm 33 ($D$)

On input $\langle M \rangle$

- Run $H$ on input $\langle M, M \rangle$.
- Output the opposite of what $H$ outputs:
  - Reject if $H$ accepts, and
  - Accept if $H$ rejects.

What happens if we run $D$ on its own description?

$$D(\langle D \rangle) = \begin{cases} \text{reject} & \text{if } D \text{ accepts } \langle D \rangle \\ \text{accept} & \text{if } D \text{ rejects } \langle D \rangle \end{cases}$$

Oh, oh...
Proving Thm 31

Suppose a TM, $H$, is a decider for $A_{TM}$. Namely,

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

Algorithm 33 ($D$)

On input $\langle M \rangle$

- Run $H$ on input $\langle M, M \rangle$.
- Output the opposite of what $H$ outputs:
  - Reject if $H$ accepts, and
  - Accept if $H$ rejects.

What happens if we run $D$ on its own description?

$$D(\langle D \rangle) = \begin{cases} \text{reject} & \text{if } D \text{ accepts } \langle D \rangle \\ \text{accept} & \text{if } D \text{ rejects } \langle D \rangle \end{cases}$$

Oh, oh...

Or, more accurately, a contradiction (to what?)
Self reference

Don’t be confused by the notion of running a machine on its own description!

Actually, you should get used to it.
Self reference

Don’t be confused by the notion of running a machine on its own description!

Actually, you should get used to it.

- Notion of self-reference comes up again and again in diverse areas.
Self reference

Don’t be confused by the notion of running a machine on its own description!

Actually, you should get used to it.

- Notion of self-reference comes up again and again in diverse areas.
- This notion of self-reference is the basic idea behind Gödel’s revolutionary result.
Self reference

Don’t be confused by the notion of running a machine on its own description!

Actually, you should get used to it.

- Notion of **self-reference** comes up again and again in diverse areas.
- This notion of self-reference is the basic idea behind Gödel’s revolutionary result.
- Compilers do this all the time . . . .
A non-enumerable language

> We already saw a non-decidable language: $A_{TM}$.
A non-enumerable language

- We already saw a non-decidable language: $A_{TM}$.
- We now display a language that is not even recursively enumerable . . .
A non-enumerable language

- We already saw a non-decidable language: $A_{TM}$.
- We now display a language that is not even recursively enumerable . . . .

Corollary 34
If $L \not\in R$, then either $L \in RE$ or $L \not\in RE$.

Proof: Assume otherwise, by Thm 22 $L$ is decidable.

Corollary 35
$A_{TM} \in RE$. 
A non-enumerable language

- We already saw a non-decidable language: $A_{TM}$.
- We now display a language that is not even recursively enumerable . . .

**Corollary 34**

*If $L \notin R$, then either $L \notin RE$ or $\overline{L} \notin RE$.***
A non-enumerable language

- We already saw a non-decidable language: $A_{TM}$.
- We now display a language that is not even recursively enumerable . . . .

**Corollary 34**

If $L \notin R$, then either $L \notin RE$ or $\overline{L} \notin RE$.

**Proof:** Assume otherwise, by Thm 22 $L$ is decidable. ♣
A non-enumerable language

- We already saw a non-decidable language: $A_{TM}$.
- We now display a language that is not even recursively enumerable . . . .

**Corollary 34**

*If $L \notin R$, then either $L \notin RE$ or $\overline{L} \notin RE$.***

Proof: Assume otherwise, by Thm 22 $L$ is decidable. ♣
A non-enumerable language

- We already saw a non-decidable language: $A_{TM}$.
- We now display a language that is not even recursively enumerable . . . .

**Corollary 34**

*If $L \notin R$, then either $L \notin RE$ or $\overline{L} \notin RE$.***

**Proof:** Assume otherwise, by Thm 22 $L$ is decidable. ♣

**Corollary 35**

*$A_{TM} \notin RE$.***
The Halting Problem

\[ H_{\text{TM}} = \{ \langle M, w \rangle : M \text{ is a TM and } M \text{ halts on input } w \} \]

**Theorem 36**

\( H_{\text{TM}} \) is undecidable.
The Halting Problem

\[ H_{\text{TM}} = \{ \langle M, w \rangle : M \text{ is a TM and } M \text{ halts on input } w \} \]

**Theorem 36**

\( H_{\text{TM}} \) is undecidable.

Proof idea:

- Similar to \( A_{\text{TM}} \).
The Halting Problem

\[ H_{TM} = \{ \langle M, w \rangle : M \text{ is a TM and } M \text{ halts on input } w \} \]

**Theorem 36**

\( H_{TM} \) is undecidable.

Proof idea:

- Similar to \( A_{TM} \).

- Alternatively, by a reduction to \( A_{TM} \) (?).
The World as we (currently) Know It

???

<table>
<thead>
<tr>
<th>co-enumerable</th>
<th>decidable</th>
<th>enumerable</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\overline{A}_{TM}$</td>
<td>$A_{DFA}$</td>
<td>$A_{TM}$</td>
</tr>
</tbody>
</table>
The World as we (currently) Know It

Are there any languages in the area marked ????
The World as we (currently) Know It

Question 37

Are there any languages in the area marked ??? ?

Yes, for instance

\[ L_3 = \{ \langle M_1 \rangle, w_1, \langle M_2 \rangle, w_2 : w_1 \in L(M_1) \land w_2 \notin L(M_2) \} \]
Claim: $L_3 = \{ \langle M_1 \rangle, w_1, \langle M_2 \rangle, w_2 : w_1 \in L(M_1) \land w_2 \notin L(M_2) \} \notin \mathcal{RE}$. 
Claim: $L_3 = \{\langle M_1 \rangle, w_1, \langle M_2 \rangle, w_2 : w_1 \in L(M_1) \land w_2 \notin L(M_2)\} \notin \mathcal{RE}$.

Proof: Assume $L_3 \in \mathcal{RE}$. Hence, $\exists$ TM $A$ with $L(A) = L_3$. 
Claim: $L_3 = \{\langle M_1\rangle, w_1, \langle M_2\rangle, w_2 : w_1 \in L(M_1) \land w_2 \notin L(M_2)\} \notin \mathcal{RE}$.

Proof: Assume $L_3 \in \mathcal{RE}$. Hence, $\exists$ TM $A$ with $L(A) = L_3$.

Algorithm 38 (TM $B$ for $\overline{A_{TM}}$)

input $\langle M, w \rangle$

1. Let $C$ be a TM s.t. $L(C) = \{c\}$
2. Run $A$ on $< C, c, M, w >$.
3. Accept, if $A$ accepts.
Claim: \( L_3 = \{ \langle M_1 \rangle, w_1, \langle M_2 \rangle, w_2 : w_1 \in L(M_1) \land w_2 \notin L(M_2) \} \notin \mathcal{RE}. \)

Proof: Assume \( L_3 \in \mathcal{RE} \). Hence, \( \exists \) TM \( A \) with \( L(A) = L_3 \).

**Algorithm 38 (TM B for \( \overline{A_{TM}} \))**

**input** \( \langle M, w \rangle \)

1. Let \( C \) be a TM s.t. \( L(C) = \{ c \} \)
2. Run \( A \) on \( \langle C, c, M, w \rangle \).
3. Accept, if \( A \) accepts.

\( B \) accepts \( \overline{A_{TM}} \), contradiction.

\( \clubsuit \)
Claim: $L_3 = \{\langle M_1 \rangle, w_1, \langle M_2 \rangle, w_2 : w_1 \in L(M_1) \land w_2 \notin L(M_2)\} \notin \text{co-R.E.}$
Claim: \( L_3 = \{ \langle M_1 \rangle, w_1, \langle M_2 \rangle, w_2 : w_1 \in L(M_1) \land w_2 \notin L(M_2) \} \notin \text{co-RE} \).

Proof: Assume \( L_3 \in \text{co-RE} \). Hence, \( \exists \) TM \( A \) with \( L(A) = \overline{L_3} \).
Claim: \( L_3 = \{ \langle M_1 \rangle, w_1, \langle M_2 \rangle, w_2 : w_1 \in L(M_1) \land w_2 \notin L(M_2) \} \notin \text{co-RE}. \)

Proof: Assume \( L_3 \in \text{co-RE} \). Hence, \( \exists \) TM \( A \) with \( L(A) = \overline{L_3} \).

**Algorithm 39 (TM \( B \) for \( \overline{A_{TM}} \))**

**input** \( \langle M, w \rangle \)

1. Let \( C \) be a TM s.t. \( L(C) = \{ c \} \) and let \( d \neq c \).
2. Run \( A \) on \( < M, w, C, d > \).
3. Accept, if \( A \) accepts.
Claim: $L_3 = \{\langle M_1 \rangle, w_1, \langle M_2 \rangle, w_2 : w_1 \in L(M_1) \land w_2 \notin L(M_2) \} \notin \text{co-RE}$. 

Proof: Assume $L_3 \in \text{co-RE}$. Hence, $\exists$ TM $A$ with $L(A) = \overline{L_3}$.

**Algorithm 39 (TM $B$ for $\overline{A_{TM}}$)**

input $\langle M, w \rangle$

1. Let $C$ be a TM s.t. $L(C) = \{c\}$ and let $d \neq c$.
2. Run $A$ on $< M, w, C, d >$.
3. Accept, if $A$ accepts.

$B$ accepts $\overline{A_{TM}}$, contradiction.

$\clubsuit$
Part VI

Beyond Enumerable and co-Enumerable
Comparing sizes of sets

- Suppose $A$ and $B$ are two sets, and we wish to compare their sizes.
- If both $A$ and $B$ are finite, we can count how many elements each of them has, and compare the numbers.
- This method does not generalize to infinite sets.
- Alternatively, we can pair the elements of $A$ and $B$. If they pair perfectly, they have equal sizes.
Correspondence

Question 40
What does it mean to say that two infinite sets are of the same size?

Answered by Georg Cantor in 1873: Pair them off.

Definition 41
A map $f: A \rightarrow B$ is a correspondence, if

▶ One-to-one: if $a_1 \neq a_2$ then $f(a_1) \neq f(a_2)$.

▶ Onto: for every $b \in B$, there is an $a \in A$ such that $f(a) = b$.

Definition 42
$A$ and $B$ are of the same size, if there is a correspondence from $A$ to $B$. 
Question 40

What does it mean to say that two infinite sets are of the same size?

Answered by Georg Cantor in 1873: Pair them off.
Correspondence

Question 40
What does it mean to say that two infinite sets are of the same size?

Answered by Georg Cantor in 1873: Pair them off.

Definition 41
A map $f : A \mapsto B$ is a correspondence, if $f$ satisfies:

- One-to-one: if $a_1 \neq a_2$ then $f(a_1) \neq f(a_2)$.
- Onto: for every $b \in B$, there is an $a \in A$ such that $f(a) = b$. 
Question 40
What does it mean to say that two infinite sets are of the same size?

Answered by Georg Cantor in 1873: Pair them off.

Definition 41
A map \( f : A \mapsto B \) is a correspondence, if \( f \) satisfies:

- One-to-one: if \( a_1 \neq a_2 \) then \( f(a_1) \neq f(a_2) \).
- Onto: for every \( b \in B \), there is an \( a \in A \) such that \( f(a) = b \).

Definition 42
\( A \) and \( B \) are of the same size, if there is a correspondence from \( A \) to \( B \).
Correspondence – Example

Claim 43
The set $\mathbb{N}$ of natural numbers has the same size as the set $\mathbb{E}$ of even numbers.

Proof: The mapping $f(i) = 2i$ is a correspondence from $\mathbb{N}$ to $\mathbb{E}$.

Remark 44
The set $\mathbb{E}$ is a proper subset of the set $\mathbb{N}$, yet they are the same size!
Correspondence – Example

Claim 43

The set $\mathbb{N}$ of natural numbers has the same size as the set $\mathbb{E}$ of even numbers.

Proof: The mapping $f(i) = 2i$ is a correspondence from $\mathbb{N}$ to $\mathbb{E}$. ♣
Correspondence – Example

Claim 43
The set \( \mathbb{N} \) of natural numbers has the same size as the set \( \mathbb{E} \) of even numbers.

Proof: The mapping \( f(i) = 2i \) is a correspondence from \( \mathbb{N} \) to \( \mathbb{E} \).

Remark 44
The set \( \mathbb{E} \) is a proper subset of the set \( \mathbb{N} \), yet they are the same size!
Countable sets

Definition 45

A set $A$ is **countable** if

- It is finite, or
- has the same size as $\mathbb{N}$. 

We have just seen that $E$ is countable.

An infinite countable set is sometimes said to have size $\aleph_0$.

Claim 46

Assuming $\exists$ a one-to-one mapping from a set $S$ to a countable set, then $S$ is countable.

Proof: Exercise.
Countable sets

Definition 45
A set $A$ is countable if

- It is finite, or
- has the same size as $\mathbb{N}$.

We have just seen that $E$ is countable.
Countable sets

**Definition 45**

A set $A$ is **countable** if

- It is finite, or
- has the same size as $\mathbb{N}$.

We have just seen that $E$ is countable.

An infinite countable set is sometimes said to have size $\aleph_0$. 
Countable sets

**Definition 45**

A set $A$ is countable if

- It is finite, or
- has the same size as $\mathbb{N}$.

We have just seen that $E$ is countable.

An infinite countable set is sometimes said to have size $\aleph_0$.

**Claim 46**

Assuming $\exists$ a one-to-one mapping from a set $S$ to a countable set, then $S$ is countable.
Countable sets

Definition 45
A set $A$ is countable if

- It is finite, or
- has the same size as $\mathbb{N}$.

We have just seen that $E$ is countable.

An infinite countable set is sometimes said to have size $\aleph_0$.

Claim 46
Assuming $\exists$ a one-to-one mapping from a set $S$ to a countable set, then $S$ is countable.

Proof: Exercise...♣
The set of all words is countable

Theorem 47

For every finite $\Sigma$, the set $\Sigma^*$ is countable.

Proof: Consider $\Sigma = \{0, 1\}$.

Define $f: \Sigma^* \mapsto \mathbb{N}$ by $f(w) = \text{bin}(1w)$, for $\text{bin}(\sigma_n, \ldots, \sigma_1) = |\Sigma| \cdot \text{bin}(\sigma_n, \ldots, \sigma_2) + \text{bin}(\sigma_1)$ (i.e., $\text{bin}(w)$ is the integer corresponding to $w$).

Therefore, $f$ is one-to-one and onto.

Why do we need the leading 1?
The set of all words is countable

**Theorem 47**

*For every finite $\Sigma$, the set $\Sigma^*$ is countable.*

**Proof:** Consider $\Sigma = \{0, 1\}$.

Define $f : \Sigma^* \mapsto \mathbb{N}$ by $f(w) = \text{bin}(1w)$, for

$$\text{bin}(\sigma_n, \ldots, \sigma_1) = |\Sigma| \cdot \text{bin}(\sigma_n, \ldots, \sigma_2) + \text{bin}(\sigma_1)$$

(i.e., $\text{bin}(w)$ is the integer corresponding to $w$).

Therefore, $f$ is one-to-one and onto. ♣
The set of all words is countable

**Theorem 47**

*For every finite \( \Sigma \), the set \( \Sigma^* \) is countable.*

**Proof:** Consider \( \Sigma = \{0, 1\} \).

Define \( f : \Sigma^* \mapsto \mathbb{N} \) by \( f(w) = \text{bin}(1w) \), for

\[
\text{bin}(\sigma_n, \ldots, \sigma_1) = |\Sigma| \cdot \text{bin}(\sigma_n, \ldots, \sigma_2) + \text{bin}(\sigma_1)
\]

(i.e., \( \text{bin}(w) \) is the integer corresponding to \( w \)).

Therefore, \( f \) is one-to-one and onto. ♣

Why do we need the leading 1?
The set of all TM’s is **countable**

**Claim 48**

The set of all TM’s is countable.
The set of all TM’s is **countable**

### Claim 48

The set of all TM’s is countable.

**Proof:**

- Each TM $M$ has an encoding as a string $\langle M \rangle$.
- Define $f : TM \mapsto \Sigma^*$ as $f(M) = \langle M \rangle$.
- Therefore $f$ is a one-to-one mapping from the set of all TMs into (but not onto) $\Sigma^*$.
- Since $\Sigma^*$ is countable, so is the set of all TMs.

♣
The set of all infinite binary strings is uncountable

**Theorem 49**

Let $\mathcal{B}$ be the set of infinite binary sequences. Then $\mathcal{B}$ is uncountable.
The set of all infinite binary strings is uncountable

**Theorem 49**

Let $\mathcal{B}$ be the set of infinite binary sequences. Then $\mathcal{B}$ is uncountable.

**Proof:** Diagonalization argument, essentially identical to the proof that $\mathbb{R}$ is uncountable.

- Assume $\mathcal{B}$ is countable using $f : \mathcal{B} \mapsto \mathbb{N}$.
- Let $b_i \in \mathcal{B}$ be the string with $f(b_i) = i$.
- Define the infinite string $w \in \mathcal{B}$, by $w_i = 1 - (b_i)_i$
- Assume $f(w) = k$. What is the value of $w_k$?

♣
The set of all languages is **uncountable**

**Theorem 50**

Let $\mathcal{L}$ be the set of all languages over $\Sigma$. Then $\exists$ correspondence from $\mathcal{L}$ to $\mathcal{B}$
The set of all languages is **uncountable**

**Theorem 50**

Let $\mathcal{L}$ be the set of all languages over $\Sigma$. Then there exists a correspondence from $\mathcal{L}$ to $\mathcal{B}$.

Hence $\mathcal{L}$ is uncountable.
The set of all languages is **uncountable**

**Theorem 50**

Let $\mathcal{L}$ be the set of all languages over $\Sigma$. Then $\exists$ correspondence from $\mathcal{L}$ to $\mathcal{B}$

Hence $\mathcal{L}$ is uncountable. **Proof:**

**Definition 51 (The characteristic function of $L$)**

Let $s_i$ be the $i$’th word of $\Sigma^*$ (in lexicographic order). Define $\chi(L) \in \mathcal{B}$ by $\chi(L)_i = 1$ if $s_i \in L$, and 0 otherwise.
The set of all languages is uncountable

**Theorem 50**

Let $L$ be the set of all languages over $\Sigma$. Then $\exists$ correspondence from $L$ to $B$

Hence $L$ is uncountable. **Proof:**

**Definition 51 (The characteristic function of $L$)**

Let $s_i$ be the $i$'th word of $\Sigma^*$ (in lexicographic order). Define $\chi(L) \in B$ by $\chi(L)_i = 1$ if $s_i \in L$, and 0 otherwise.

**Example 52**

\[
\begin{align*}
\Sigma^* &= \{ \varepsilon, 0, 1, 00, 01, 10, 11, 000 \ldots \} \\
L &= \{ 0, 00, 01, 000 \ldots \} \\
\chi(L) &= \{ 0, 1, 0, 1, 1, 0, 0, 1 \ldots \}
\end{align*}
\]
The set of all languages is uncountable

**Theorem 50**

Let \( \mathcal{L} \) be the set of all languages over \( \Sigma \). Then \( \exists \) correspondence from \( \mathcal{L} \) to \( B \)

Hence \( \mathcal{L} \) is uncountable. **Proof:**

**Definition 51 (The characteristic function of \( \mathcal{L} \))**

Let \( s_i \) be the \( i \)'th word of \( \Sigma^* \) (in lexicographic order). Define \( \chi(L) \in B \) by \( \chi(L)_i = 1 \) if \( s_i \in L \), and 0 otherwise.

**Example 52**

\[
\begin{align*}
\Sigma^* &= \{ \varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots \} \\
L &= \{ 0, 00, 01, 000, \ldots \} \\
\chi(L) &= \{ 0, 1, 0, 1, 1, 0, 0, 1, \ldots \}
\end{align*}
\]

**Claim 53**

\( \chi: \mathcal{L} \mapsto B \) is one-to-one and onto.
The set of all languages is uncountable

Theorem 50

Let \( \mathcal{L} \) be the set of all languages over \( \Sigma \). Then \( \exists \) correspondence from \( \mathcal{L} \) to \( \mathcal{B} \).

Hence \( \mathcal{L} \) is uncountable. Proof:

Definition 51 (The characteristic function of \( L \))

Let \( s_i \) be the \( i \)'th word of \( \Sigma^* \) (in lexicographic order). Define \( \chi(L) \in \mathcal{B} \) by \( \chi(L)_i = 1 \) if \( s_i \in L \), and 0 otherwise.

Example 52

\[
\begin{align*}
\Sigma^* &= \{ \varepsilon, 0, 1, 00, 01, 10, 11, 000 \ldots \} \\
L &= \{ 0, 00, 01, 000 \ldots \} \\
\chi(L) &= \{ 0, 1, 0, 1, 1, 0, 0, 1 \ldots \}
\end{align*}
\]

Claim 53

\( \chi : \mathcal{L} \mapsto \mathcal{B} \) is one-to-one and onto.

Hence, \( \chi \) is a correspondence, yielding that \( \mathcal{L} \) is uncountable.
TM vs. Languages

- The set of all TM is countable.

- The set of all languages is uncountable.

- Therefore, there are languages outside \( \text{RE} \) (why?).

- Moreover, there are languages outside \( \text{RE} \cup \text{co-RE} \) (why?).

- This is an existential proof – it does not explicitly show any such language.
TMs vs. Languages

- The set of all TM is countable.
- The set of all languages is uncountable.

Therefore, there are languages outside \( \text{RE} \) (why?).

Moreover, there are languages outside \( \text{RE} \cup \text{co}-\text{RE} \) (why?).

This is an existential proof – it does not explicitly show any such language.
TMs vs. Languages

- The set of all TM is countable.
- The set of all languages is uncountable.
- Therefore, there are languages outside $\text{RE}$ (why?).

Moreover, there are languages outside $\text{RE} \cup \text{co-RE}$ (why?). This is an existential proof – it does not explicitly show any such language.
TMs vs. Languages

- The set of all TM is countable.
- The set of all languages is uncountable.
- Therefore, there are languages outside $\mathcal{RE}$ (why?).
- Moreover, there are languages outside $\mathcal{RE} \cup \text{co-\mathcal{RE}}$ (why?).
TMs vs. Languages

- The set of all TM is countable.
- The set of all languages is uncountable.
- Therefore, there are languages outside $\mathsf{RE}$ (why?).
- Moreover, there are languages outside $\mathsf{RE} \cup \mathsf{co-RE}$ (why?).
- This is an existential proof – it does not explicitly show any such language.