Clear

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

Example 2.1 (Hilbert's Tenth Problem): "Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers." (→ Wikipedia)

Question

How can we model the notion of an algorithm?

Answer

With Turing machines.
A Model for Computation

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

Example 2.1 (Hilbert’s Tenth Problem):
“Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers.”
(→ Wikipedia)
A Model for Computation

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

Example 2.1 (Hilbert’s Tenth Problem):
“Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers.”
(→ Wikipedia)

Question
How can we model the notion of an algorithm?
A Model for Computation

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

Example 2.1 (Hilbert’s Tenth Problem):
“Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers.”
(→ Wikipedia)

Question
How can we model the notion of an algorithm?

Answer
With Turing machines.
Turing Machines

Let us fix a blank symbol \(\_\).
Turing Machines

Let us fix a blank symbol $\bl$.

**Definition 2.2:** A (deterministic) Turing Machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}} \rangle$ consists of

- a finite set $Q$ of states,
- an input alphabet $\Sigma$ not containing $\bl$,
- a tape alphabet $\Gamma$ such that $\Gamma \supseteq \Sigma \cup \{\bl\}$.
- a transition function $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- an initial state $q_0 \in Q$,
- an accepting state $q_{\text{accept}} \in Q$, and
- an rejecting state $q_{\text{reject}} \in Q$ such that $q_{\text{accept}} \neq q_{\text{reject}}$. 
Example 2.3:

- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over $\Gamma$, followed by an infinite sequence of $\_$.  
- The head of the machine is at exactly one position of the tape.  
- The head can read only one symbol at a time.  
- The head moves and writes according to the transition function $\delta$; the current state also changes accordingly.  
- The head will stay put when attempting to cross the left tape end.
Example 2.3:

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

[[Diagram of Turing machine tape with states and symbols]]
Example 2.3:

- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over $\Gamma$, followed by an infinite sequence of $\sqcup$.
- The head of the machine is at exactly one position of the tape.
Example 2.3:

- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over $\Gamma$, followed by an infinite sequence of $\bot$.
- The head of the machine is at exactly one position of the tape.
Example 2.3:

- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over $\Gamma$, followed by an infinite sequence of $\downarrow$.
- The head of the machine is at exactly one position of the tape.
- The head can read only one symbol at a time.
Example 2.3:

- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over $\Gamma$, followed by an infinite sequence of $\sqcup$.
- The head of the machine is at exactly one position of the tape.
- The head can read only one symbol at a time.
- The head moves and writes according to the transition function $\delta$; the current state also changes accordingly.
Example 2.3:

- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over $\Gamma$, followed by an infinite sequence of $\cdot$.
- The head of the machine is at exactly one position of the tape.
- The head can read only one symbol at a time.
- The head moves and writes according to the transition function $\delta$; the current state also changes accordingly.
Example 2.3:

\[ \delta(q_1, a) = (q_2, b, L) \]

- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over \( \Gamma \), followed by an infinite sequence of \( \_ \).
- The head of the machine is at exactly one position of the tape.
- The head can read only one symbol at a time.
- The head moves and writes according to the transition function \( \delta \); the current state also changes accordingly.
Example 2.3:

\[ \delta(q_1, a) = (q_2, b, \Lambda) \]

- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over \( \Gamma \), followed by an infinite sequence of \( \Lambda \).
- The head of the machine is at exactly one position of the tape.
- The head can read only one symbol at a time.
- The head moves and writes according to the transition function \( \delta \); the current state also changes accordingly.
Example 2.3:

\[ \delta(q_1, a) = (q_2, b, L) \]

- The tape is bounded on the left, but unbounded on the right; the content of the tape is a finite word over \( \Gamma \), followed by an infinite sequence of \( \_ \).
- The head of the machine is at exactly one position of the tape.
- The head can read only one symbol at a time.
- The head moves and writes according to the transition function \( \delta \); the current state also changes accordingly.
- The head will stay put when attempting to cross the left tape end.
Observation: to describe the current step of a computation of a TM it is enough to know

- the content of the tape,
- the current state, and
- the position of the head
Observation: to describe the current step of a computation of a TM it is enough to know

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

**Definition 2.4:** A configuration of a TM \( M \) is a word \( uqv \) such that

- \( q \in Q \),
- \( uv \in \Gamma^* \)
Observation: to describe the current step of a computation of a TM it is enough to know

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

**Definition 2.4:** A configuration of a TM $M$ is a word $uqv$ such that

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

Some special configurations:

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

- \( C \vdash_M C' \) only if \( C' \) can be reached from \( C \) by one computation step of \( M \);
- \( C \vdash_M^* C' \) only if \( C' \) can be reached from \( C \) in a finite number of computation steps of \( M \).
We write
- $C ⊢_M C'$ only if $C'$ can be reached from $C$ by one computation step of $M$;
- $C ⊢^*_M C'$ only if $C'$ can be reached from $C$ in a finite number of computation steps of $M$.

We say that $M$ halts on input $w$ if and only if there is a finite sequence of configurations

$$C_0 ⊢_M C_1 ⊢_M \cdots ⊢_M C_\ell$$

such that $C_0$ is the start configuration of $M$ on input $w$ and $C_\ell$ is an accepting or rejecting configuration. Otherwise $M$ loops on input $w$. 
Computation

We write

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

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

$$C_0 \vdash_M C_1 \vdash_M \cdots \vdash_M C_\ell$$

such that $C_0$ is the start configuration of $M$ on input $w$ and $C_\ell$ is an accepting or rejecting configuration. Otherwise $M$ **loops** on input $w$.

We say that $M$ **accepts** the input $w$ only if $M$ halts on input $w$ with an accepting configuration.
**Definition 2.5:** Let $M$ be a Turing machine with input alphabet $\Sigma$. The language accepted by $M$ is the set

$$L(M) := \{ w \in \Sigma^* \mid M \text{ accepts } w \}.$$
Definition 2.5: Let $M$ be a Turing machine with input alphabet $\Sigma$. The language accepted by $M$ is the set

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

A language $L \subseteq \Sigma^*$ is called Turing-recognisable (recursively enumerable) if and only if there exists a Turing machine $M$ with input alphabet $\Sigma^*$ such that $L = L(M)$. In this case we say that $M$ recognises $L$. 
Definition 2.5: Let $\mathcal{M}$ be a Turing machine with input alphabet $\Sigma$. The language accepted by $\mathcal{M}$ is the set

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

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

A language $L \subseteq \Sigma^*$ is called Turing-decidable (decidable, recursive) if and only if there exists a Turing machine $\mathcal{M}$ such that $L = L(\mathcal{M})$ and $\mathcal{M}$ halts on every input. In this case we say that $\mathcal{M}$ decides $L$. 
Claim 2.6: The language $L := \{ a^{2^n} \mid n \geq 0 \}$ is decidable.
Claim 2.6: The language $L := \{ a^{2^n} \mid n \geq 0 \}$ is decidable.

Proof: A Turing machine $M$ that decides $L$ is

$M :=$ On input $w$, where $w$ is a string

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

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

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

and $\delta$ is given by

![Diagram showing the states and transitions of the automaton.](image-url)
Observation

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

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

**Example 2.7 (Graph-Connectedness):** The question whether a graph is connected or not can be seen as the **word problem** of the following language

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

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

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

\textbf{Example 2.7 (Graph-Connectedness):} The question whether a graph is connected or not can be seen as the \textbf{word problem} of the following language

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

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

\textbf{Notation 2.8:} The encoding of objects \( O_1, \ldots, O_n \) we denote by \( \langle O_1, \ldots, O_n \rangle \).
The Church-Turing Thesis

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

- $\lambda$-calculus
- while-programs
- $\mu$-recursive functions
- Random-Access Machines
- ...
The Church-Turing Thesis

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

- $\lambda$-calculus
- while-programs
- $\mu$-recursive functions
- Random-Access Machines
- ...

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

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

(\(\rightarrow\) Wikipedia: Church-Turing Thesis)
Variations of Turing-Machines

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

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

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

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

$k$-tape Turing machines are a variant of Turing machines that have $k$ tapes.
$k$-tape Turing machines are a variant of Turing machines that have $k$ tapes.
Definition 2.9: Let $k \in \mathbb{N}$. Then a (deterministic) $k$-tape Turing machine is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where

- $Q, \Sigma, \Gamma, q_0, q_{\text{accept}}, q_{\text{reject}}$ are as for TMs
- $\delta$ is a transition function for $k$ tapes, i.e.,

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, N\}^k$$
Definition 2.9: Let $k \in \mathbb{N}$. Then a (deterministic) $k$-tape Turing machine is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where

- $Q$, $\Sigma$, $\Gamma$, $q_0$, $q_{\text{accept}}$, $q_{\text{reject}}$ are as for TMs
- $\delta$ is a transition function for $k$ tapes, i.e.,

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

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

**Definition 2.9:** Let $k \in \mathbb{N}$. Then a (deterministic) $k$-tape Turing machine is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where

- $Q$, $\Sigma$, $\Gamma$, $q_0$, $q_{\text{accept}}$, $q_{\text{reject}}$ are as for TMs
- $\delta$ is a transition function for $k$ tapes, i.e.,

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, N\}^k$$

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

The notions of a **configuration** and of the **language accepted by $M$** are defined analogously to the single-tape case.
**Theorem 2.10:** Every multi-tape Turing machine has an equivalent single-tape Turing machine.
Theorem 2.10: Every multi-tape Turing machine has an equivalent single-tape Turing machine.

Proof: Let $M$ be a $k$-tape Turing machine. Simulate $M$ with a single-tape TM $S$ by...
**Theorem 2.10:** Every multi-tape Turing machine has an equivalent single-tape Turing machine.

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

- keeping the content of all $k$ tapes on a single tape, separated by #
Theorem 2.10: Every multi-tape Turing machine has an equivalent single-tape Turing machine.

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

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

**Theorem 2.10:** Every multi-tape Turing machine has an equivalent single-tape Turing machine.

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

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

![Diagram of multi-tape and single-tape Turing machines]
Multi-Tape Turing Machines

\[ S := \text{On input } w = w_1 \ldots w_n \]
Multi-Tape Turing Machines

\[ S := \text{On input } w = w_1 \ldots w_n \]

- Format the tape to contain the word

\[ \#w_1w_2 \ldots w_n\#\#\#\#\ldots\# \]
Multi-Tape Turing Machines

\[ S \ := \text{On input } w = w_1 \ldots w_n \]

- Format the tape to contain the word

\[
\# w_1 w_2 \ldots w_n \# \# \# \ldots \#
\]

- Scan the tape from the first # to the \((k + 1)\)-th # to determine the symbols below the markers.
Multi-Tape Turing Machines

\[ S := \text{On input } w = w_1 \ldots w_n \]

- Format the tape to contain the word

\[ \# w_1 w_2 \ldots w_n \# \# \# \ldots \# \]

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

- Update all tapes according to \(M\)'s transition function with a second pass over the tape; if any head of \(M\) moves to some previously unread portion of its tape, insert a blank symbol at the corresponding position and shift the right tape contents by one cell.
Multi-Tape Turing Machines

$S := \text{On input } w = w_1 \ldots w_n$

- Format the tape to contain the word

  $$\#w_1w_2 \ldots w_n\#\#\#\#\ldots\#$$

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

- Update all tapes according to $M$'s transition function with a second pass over the tape; if any head of $M$ moves to some previously unread portion of its tape, insert a blank symbol at the corresponding position and shift the right tape contents by one cell.

- Repeat until the accepting or rejecting state is reached.
Multi-Tape Turing Machines

\[ S := \text{On input } w = w_1 \ldots w_n \]

- Format the tape to contain the word

\[ \# w_1 w_2 \ldots w_n \# \# \# \ldots \# \]

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

- Update all tapes according to \(M\)'s transition function with a second pass over the tape; if any head of \(M\) moves to some previously unread portion of its tape, insert a blank symbol at the corresponding position and shift the right tape contents by one cell.

- Repeat until the accepting or rejecting state is reached.
Nondeterministic Turing Machines

Goal
Allow transitions to be **nondeterministic**.
Nondeterministic Turing Machines

Goal
Allow transitions to be **nondeterministic**.

Approach
Change transition function from

\[ \delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\} \]

to

\[ \delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L, R\}} \].
Nondeterministic Turing Machines

Goal
Allow transitions to be **nondeterministic**.

Approach
Change transition function from

\[
\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{ L, R \}
\]

to

\[
\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{ L, R \}}.
\]

The notions of **accepting** and **rejecting computations** are defined accordingly.

**Note**: there may be more than one or no computation of a nondeterministic TM on a given input.
Nondeterministic Turing Machines

Goal
Allow transitions to be **nondeterministic**.

Approach
Change transition function from

\[ \delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} \]

to

\[ \delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}} \].

The notions of **accepting** and **rejecting computations** are defined accordingly.

**Note**: there may be more than one or no computation of a nondeterministic TM on a given input.

A nondeterministic TM $M$ **accepts** an input $w$ if and only if there exists some accepting computation of $M$ on input $w$. 
Theorem 2.11: Every nondeterministic TM has an equivalent deterministic TM.
Theorem 2.11: Every nondeterministic TM has an equivalent deterministic TM.

Proof: Let $N$ be a nondeterministic TM. We construct a deterministic TM $D$ that is equivalent to $N$, i.e., $L(N) = L(D)$. 
Theorem 2.11: Every nondeterministic TM has an equivalent deterministic TM.

Proof: Let $N$ be a nondeterministic TM. We construct a deterministic TM $D$ that is equivalent to $N$, i.e., $L(N) = L(D)$.

Idea
Theorem 2.11: Every nondeterministic TM has an equivalent deterministic TM.

Proof: Let $N$ be a nondeterministic TM. We construct a deterministic TM $D$ that is equivalent to $N$, i.e., $L(N) = L(D)$.

Idea
- $D$ deterministically traverses in breath-first order the tree of configuration of $N$, where each branch represents a different possibility for $N$ to continue.
**Theorem 2.11**: Every nondeterministic TM has an equivalent deterministic TM.

**Proof**: Let \( N \) be a nondeterministic TM. We construct a deterministic TM \( D \) that is equivalent to \( N \), i.e., \( L(N) = L(D) \).

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

Sketch of $D$:

Let $b$ be the maximal number of choices in $\delta$, i.e.,
$$b = \max \{|\delta(q, x)| : q \in Q, x \in \Gamma\}.$$
Sketch of $D$:

Let $b$ be the maximal number of choices in $\delta$, i.e.,

$$b := \max\{|\delta(q, x)| \mid q \in Q, x \in \Gamma\}.$$
Nondeterministic Turing Machines

\(D\) works as follows:

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

\(\square\)
$D$ works as follows:

1. Start: input tape contains input $w$, simulation and address tape empty.
$D$ works as follows:

1. Start: input tape contains input $w$, simulation and address tape empty
2. Copy $w$ to the simulation tape and initialise the address tape with 0.
$D$ works as follows:

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

$D$ works as follows:

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

$D$ works as follows:

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

(2) Copy $w$ to the simulation tape and initialise the address tape with 0.

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

1. Start: input tape contains input $w$, simulation and address tape empty
2. Copy $w$ to the simulation tape and initialise the address tape with 0.
3. Simulate one finite computation of $N$ on $w$ on the simulation tape.
   - Interpret the address tape as a list of choices to make during this computation.
   - If a choice is invalid, abort simulation.
   - If an accepting configuration is reached at the end of the simulation, accept.

\[ \text{□} \]
$D$ works as follows:

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

$D$ works as follows:

1. Start: input tape contains input $w$, simulation and address tape empty
2. Copy $w$ to the simulation tape and initialise the address tape with 0.
3. Simulate one finite computation of $N$ on $w$ on the simulation tape.
   - Interpret the address tape as a list of choices to make during this computation.
   - If a choice is invalid, abort simulation.
   - If an accepting configuration is reached at the end of the simulation, accept.
4. Increment the content of the address tape, considered as a number in base $b$, by 1.
   Go to step 2.
Definition 2.12: A multi-tape Turing machine $M$ is an enumerator if

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

We define the language generated by $M$ to be the set $G(M)$ of all words that eventually appear between two consecutive # on the output tape of $M$ when started on the empty word as input.
**Definition 2.12:** A multi-tape Turing machine $M$ is an **enumerator** if

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

We define the **language generated by** $M$ to be the set $G(M)$ of all words that eventually appear between two consecutive $#$ on the output tape of $M$ when started on the empty word as input.
Theorem 2.13: A language $L$ is Turing-recognisable if and only if there exists some enumerator $E$ such that $G(E) = L$. 
Theorem 2.13: A language $L$ is Turing-recognisable if and only if there exists some enumerator $E$ such that $G(E) = L$.

Proof: Let $E$ be an enumerator for $L$. Then the following TM accepts $L$:

$M :=$ On input $w$

- Simulate $E$ on the empty input. Compare every string output by $E$ with $w$
- If $w$ appears in the output of $E$, accept
Let $L = L(M)$ for some TM $M$, and let $s_1, s_2, \ldots$ be an enumeration of $\Sigma^*$. 
Enumerators

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

\[
E := \text{Ignore the input.}
\]

- Repeat for \( i = 1, 2, 3, \ldots \)
  - Run \( M \) for \( i \) steps on each input \( s_1, s_2, \ldots, s_i \)
  - If any computation accepts, print the corresponding \( s_j \) followed by \#
Let \( L = L(M) \) for some TM \( M \), and let \( s_1, s_2, \ldots \) be an enumeration of \( \Sigma^* \). Then the following enumerator \( E \) enumerates \( L \):

\[
E := \text{Ignore the input.}
\]

- Repeat for \( i = 1, 2, 3, \ldots \)
  - Run \( M \) for \( i \) steps on each input \( s_1, s_2, \ldots, s_i \)
  - If any computation accepts, print the corresponding \( s_j \) followed by #
Theorem 2.15: A language $L$ is decidable if and only if there exists an enumerator for $L$ that outputs exactly the words of $L$ in some order of non-decreasing length.
Theorem 2.15: A language $L$ is decidable if and only if there exists an enumerator for $L$ that outputs exactly the words of $L$ in some order of non-decreasing length.

Proof: Suppose $L$ to be decidable, and let $M$ be a TM that decides $L$. 
Theorem 2.15: A language $L$ is decidable if and only if there exists an enumerator for $L$ that outputs exactly the words of $L$ in some order of non-decreasing length.

Proof: Suppose $L$ to be decidable, and let $M$ be a TM that decides $L$.

- Define a TM $M'$ that generates, on some scratch tape, all words over $\Sigma$ in some order of non-decreasing length. (Exercise!)
Theorem 2.15: A language $L$ is decidable if and only if there exists an enumerator for $L$ that outputs exactly the words of $L$ in some order of non-decreasing length.

Proof: Suppose $L$ to be decidable, and let $M$ be a TM that decides $L$.

- Define a TM $M'$ that generates, on some scratch tape, all words over $\Sigma$ in some order of non-decreasing length. (Exercise!)
- For each word $w$ thus generated, simulate $M$ on $w_i$. If $M$ accepts $w$, then $M'$ prints $w$ followed by #.
Theorem 2.15: A language \( L \) is decidable if and only if there exists an enumerator for \( L \) that outputs exactly the words of \( L \) in some order of non-decreasing length.

Proof: Suppose \( L \) to be decidable, and let \( M \) be a TM that decides \( L \).

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

Then \( M' \) enumerates exactly the words of \( L \) in some order of non-decreasing length.
Now suppose L can be enumerated by some TM E in some order of non-decreasing length.
Enumerators

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

- If $L$ is finite, then $L$ is accepted by a finite automaton.
Now suppose \( L \) can be enumerated by some TM \( E \) in some order of non-decreasing length.

- If \( L \) is finite, then \( L \) is accepted by a finite automaton.
- If \( L \) is infinite, then we define a decider \( M \) for it as follows.
Now suppose $L$ can be enumerated by some TM $E$ in some order of non-decreasing length.

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

\[
M := \text{On input } w \\
\quad - \text{Simulate } E \text{ until it either outputs } w \text{ or some word longer than } w \\
\quad - \text{If } E \text{ outputs } w, \text{ then accept, else reject.}
\]
Enumerators

Now suppose \( L \) can be enumerated by some TM \( E \) in some order of non-decreasing length.

- If \( L \) is finite, then \( L \) is accepted by a finite automaton.
- If \( L \) is infinite, then we define a decider \( M \) for it as follows.

\[
M := \text{On input } w \\
- \text{Simulate } E \text{ until it either outputs } w \text{ or some word longer than } w \\
- \text{If } E \text{ outputs } w, \text{ then accept, else reject.}
\]

**Observation:** since \( L \) is infinite, for each \( w \in \Sigma^* \) the TM \( E \) will eventually generate \( w \) or some word longer than \( w \).
Now suppose $L$ can be enumerated by some TM $\mathcal{E}$ in some order of non-decreasing length.

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

\[ M := \text{On input } w \]

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

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

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

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

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

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

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

Turing Machines are a simple model of computation

Recognisable (semi-decidable) = recursively enumerable

Decidable = computable = recursive

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

What’s next?

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