In this lecture, we outline the Turing model of computation and show that there exist non-computable functions. We present a framework for studying the intrinsic complexity of a problem (abstracted from any particular algorithm for solving it) in terms of minimal time- and space-resources required for deterministic and non-deterministic Turing machines to solve it. We present some basic results on the relationships between these complexity classes. The lecture three parts:

- Turing machines;
- time- and space-bounds;
- a separation result.
Outline

Turing machines

Time and space

A separation result

Summary
Turing machines

Alan Turing

Time and space

Alonzo Church

A separation result

Kurt Gödel

Summary
From the point of view of the rest of this course, an *algorithm* is a *multi-tape Turing Machine*:

- We think of Tape 1 as the *input*, Tape $K$ as the *output*, and Tapes 2–($K - 1$) as *work tapes*. 
• Formally, a Turing Machine is a quintuple

\[ M = \langle K, \Sigma, Q, q_0, T \rangle, \]

where

- \( K \geq 2 \) (number of tapes)
- \( \Sigma \) is a non-empty, finite set (alphabet)
- \( Q \) is a non-empty, finite set (set of states)
- \( q_0 \in Q \) (initial state)
- \( T \) is a set of transitions (for \( K \), \( \Sigma \) and \( Q \))—see below.

• A symbol is an element of \( \Sigma \cup \{\square, \triangleright\} \)
  - We pronounce \( \square \) as blank and \( \triangleright \) as start.
• A transition (for \( K \), \( \Sigma \) and \( Q \)) is a quintuple

\[
\langle p, \bar{s}, q, \bar{t}, \bar{d} \rangle
\]

where

• \( p \in Q \) and \( q \in Q \)
• \( \bar{s} \) and \( \bar{t} \) are \( K \)-tuples of symbols
• \( \bar{d} \) is a \( K \)-tuple whose elements from \{left, right, stay\}

• It has the informal meaning:

*If you are in state \( p \), and the symbols written on the squares currently being scanned on the \( K \) tapes are given by \( \bar{s} \), then set the new state to be \( q \), write the symbols of \( \bar{t} \) on the \( K \) tapes and move the heads as directed by \( \bar{d} \).*
• We insist that
  • the tape never moves left past $\triangleright$, 
  • Tape 1 is read-only (input) and Tape K write-only (output) . . .
• $M$ is deterministic if, for every $p$ and every $\bar{s}$, there is at most one transition $\langle p, \bar{s}, q, \bar{t}, d \rangle$. 
• **A configuration** of $M$: a $K$-tuple of strings

$\triangleright, S_1, 1, \ldots, S_k, i-1, q, S_k, i, \ldots, S_k, n(k)$.

representing the situation in which the $k$th tape of $M$ reads $\triangleright, s_k, 1, \ldots, s_k, n(k)$, the head is over square $i$, and the current state is $q$ (same for all $K$ strings).

• **A run** of machine $M$ on input $x$ is a sequence of configurations (finite or infinite) in which successive configurations conform to some transition in $T$.

• Just to be clear: $M$ must make some transition if one is available.

• The run is **terminating** if it is finite. In this case, and for deterministic $M$, we write $M \downarrow x$; otherwise $M \uparrow x$. 
Definition
Let $M$ be a deterministic Turing machine over alphabet $\Sigma$, and let $x \in \Sigma^*$. If $M \downarrow x$, then the output tape of $M$ will contain a definite string $y \in \Sigma^*$; and we can define the partial function $f_M : \Sigma^* \rightarrow \Sigma^*$ as follows.

$$f_M(x) = \begin{cases} 
 y & \text{if } M \downarrow x \\
 \text{undefined} & \text{otherwise}
\end{cases}$$

We say that $M$ computes the function $f_M$. A partial function $f : \Sigma^* \rightarrow \Sigma^*$ is computable if it is computed by some (deterministic) Turing machine.
An important feature of Turing machines is that they are \textit{finite} objects of the form $\langle K, \Sigma, Q, q_0, T \rangle$.

As such, they can be encoded in some (actually: in any) alphabet.

The code of a TM can then be used as input to another TM.

There exists a \textit{universal} Turing machine:

\textbf{Theorem}

\textit{Fix some alphabet $\Sigma$. There exists a Turing machine $U$ with the following property. For any Turing Machine $M$ with alphabet $\Sigma$, and any strings $x, y \in \Sigma^*$, $U$ has a terminating run on input $(M; x)$ leaving $y$ on the output tape if and only if $M$ has a terminating run on input $x$ leaving $y$ on the output tape; moreover, $U$ has a non-terminating run on input $(M; x)$ if and only if $M$ has a non-terminating run on input $x$.}
Definition (Acceptance and recognition)

Let $M$ be a Turing machine over alphabet $\Sigma$, and let $x \in \Sigma^*$. We say $M$ accepts $x$ if $M$ has a halting run on input $x$, with the first head over the leftmost square.

(Alert: texts differ on the precise definitions here.)

The set of strings accepted by $M$ is called the language recognized by $M$.

Theorem

If a language is recognized by some Turing machine $M$, then it is recognized by some deterministic Turing Machine $M'$.
• A language is recursively enumerable (r.e.) if there is a (deterministic) Turing Machine which recognizes it.
• A language $L$ over alphabet $\Sigma$ is co-recursively enumerable (co-r.e.) if $\Sigma^* \setminus L$ is r.e.
• A language is recursive if there is a deterministic Turing Machine $M$ recognizing it, such that $M$ always halts.
• As an exercise, prove the following:
  
  A language is recursive if and only if it is both r.e. and co-r.e.

• Sometimes, we use the vocabulary of problems and decidability
  • language $\leftrightarrow$ problem
  • recursive $\leftrightarrow$ decidable
• Example: the problem of determining whether an integer is prime is the language
\( \{ x \in \{0, 1\}^* \mid x \text{ denotes a prime number} \} \).

• Consider the problem of determining whether a formula of propositional logic is satisfiable

\[
p_0 \land (p_1 \rightarrow p_{10})
\]

\[
\neg((p_0 \land p_1) \lor (p_0 \land \neg p_1) \lor (\neg p_0 \land p_1) \lor (\neg p_0 \land \neg p_1))
\]

This problem is the language language
\( \{ x \in \{ p, 0, 1, \land, (, ), \lor, \neg, \rightarrow \}^* \mid x \text{ is a wff and satisfiable} \} \).

• Both these problems are decidable.
• It is easy to see from the above that a problem $\mathcal{P} \subseteq \Sigma^*$ is decidable if and only if the total function

$$f_P : \Sigma^* \to \{\text{Yes, No}\}$$

$$f_P(x) = \begin{cases} 
\text{Yes} & \text{if } x \in \mathcal{P}; \\
\text{No} & \text{otherwise.}
\end{cases}$$

is computable by a deterministic Turing machine.

• Typically, we present problems in the form

**PROBLEM NAME**
Given: a string $x$ (coding some object we are interested in);
Return: Yes if $x$ has some property $\mathcal{P}$, and No otherwise.
Definition

The *Halting problem* is the following problem:

**HALTING**

Given: a pair of strings \( m, \ x \);
Return: Yes if \( m \) is the code of a deterministic Turing Machine, \( M \), and \( x \) a string in the alphabet of \( M \), such that \( M \downarrow x \);
No otherwise.

Theorem (Turing, 1936)

*The Halting problem is not decidable.*
**Proof.**

Suppose $H$ is a deterministic TM such that, for every deterministic TM $M$ with code $m$, and every string $x$ in the alphabet of $M$, $H$ outputs Yes on input $(m; x)$ if $M \downarrow x$, and No otherwise. Let be $H^*$ as below, with code $h^*$.

![Diagram](image)

What happens if $H^*$ is given input $h^*$? The embedded $H$ receives input $h^*; h^*$. Hence:

$$H^* \downarrow h^* \Rightarrow H^* \uparrow h^*$$

$$H^* \uparrow h^* \Rightarrow H^* \downarrow h^*$$
Outline

Turing machines

Time and space

A separation result

Summary
• In these lectures, we are interested in the resources required by Turing machines to recognize languages (= decide problems).
• There are two main types of machine:
  • deterministic
  • non-deterministic
• There are two main types of resource:
  • time
  • space
• Warning: this four-way classification is not meant to exhaust the possible types of complexity analysis!
• There now follow some rather dreary definitions . . .
Definition
Let $M$ be a Turing machine with alphabet $\Sigma$, and let $g : \mathbb{N} \to \mathbb{N}$. We say $M$ **runs in time** $g$ if, for all but finitely many strings $x \in \Sigma^*$, any run of $M$ on input $x$ halts within at most $g(|x|)$ steps. Similarly, $M$ **runs in space** $g$ if $M$ always terminates and, for all but finitely many strings $x \in \Sigma^*$, any run of $M$ on input $x$ uses at most $g(|x|)$ squares on any of its work-tapes.
• Thus, it makes sense to say, for example, that a given Turing machine runs in time (or space) $n^2$, or $3n^3 - 13n + 42$.

• Usually, however, we are not interested in the exact running times of this or that TM, since these measures tell us little about the problem at hand.

• Suppose $M$ is a Turing machine running in time/space $g$, and let $c > 0$.

• Provided $g$ is moderately fast-growing, there exists a TM $M'$, running in time $c \cdot g$, halting exactly when $M$ does, and writing the same results on its output tape. (“Linear speed-up”.)

• A similar result holds for space bounds.
• This allows us to define the intrinsic complexity of a given language (problem) $L$ in terms of the time- and space bounds of any Turing machine that recognizes (decides) it.

• We begin with non-deterministic TMs:

**Definition**
Let $L$ be a language over some alphabet, and let $G$ be a set of functions from $\mathbb{N}$ to $\mathbb{N}$. We say that $L$ is in $\text{NTime}(G)$ (or $\text{NSpace}(G)$) if there exists a Turing machine $M$ recognizing $L$ and a function $g \in G$, such that $M$ runs in time (respectively, space) $g$.

If $G = \{g\}$, we write $\text{NTime}(g)$ instead of $\text{NTime}(\{g\})$, and similarly for space.
• When talking about the complexity of problems, we typically consider fairly large classes of functions:

\[
\begin{align*}
P & = \{n^c \mid c > 0\} \\
E & = \{2^{n^c} \mid c > 0\} \\
E_2 & = \{2^{2^{n^c}} \mid c > 0\} \\
E_k & = \left\{2^{2^{\cdots^{n^c}}} \right\}^k \text{ times} | c > 0
\end{align*}
\]

• A function \( g : \mathbb{N} \to \mathbb{N} \) which is in \( E_k \) for some \( k \) is said to be elementary. (We know all about these from an earlier lecture.)
• Thus we have the following non-deterministic complexity classes.

• The usual names are on the left-hand sides of the equations, their meanings on the right-hand sides.

\[
\begin{align*}
\text{NPTIME} & = \text{NTIME}(P) \\
\text{NExpTime} & = \text{NTIME}(E) \\
k-\text{NExpTime} & = \text{NTIME}(E_k) \\
\text{NLogSpace} & = \text{NSpace}(\log n) \\
\text{NPSPACE} & = \text{NSpace}(P) \\
\text{NExpSpace} & = \text{NSpace}(E) \\
k-\text{NExpSpace} & = \text{NSpace}(E_k).
\end{align*}
\]
• We can also restrict attention to deterministic TMs:

**Definition**

Let $L$ be a language over some alphabet, and let $G$ be a set of functions from $\mathbb{N}$ to $\mathbb{N}$. We say that $L$ is in $\text{TIME}(G)$ (or $\text{SPACE}(G)$) if there exists a deterministic Turing machine $M$ recognizing $L$ and a function $g \in G$, such that $M$ runs in time (respectively, space) $g$.

If $G = \{g\}$, we write $\text{TIME}(g)$ instead of $\text{TIME}(\{g\})$, and similarly for space.
This yields the deterministic complexity classes:

- $\text{PTime} = \text{Time}(P)$
- $\text{ExpTime} = \text{Time}(E)$
- $k-$\text{ExpTime} = $\text{Time}(E_k)$
- $\text{LogSpace} = \text{Space}(\log n)$
- $\text{PSPACE} = \text{Space}(P)$
- $\text{ExpSpace} = \text{Space}(E)$
- $k-$\text{ExpSpace} = $\text{Space}(E_k)$.
Thus, we showed in earlier lectures that
- DIRECTED GRAPH CYCLICITY
- GRAPH CONNECTEDNESS
- PERFECT MATCHING

are all in $P_{\text{Time}}$. (Recall that the last of these is not so obvious: the naïve algorithm takes exponential time.)

Two famous problems that are in $P_{\text{Time}}$, but not obviously so, are
- LINEAR PROGRAMMING FEASIBILITY (Khachiyan, 1979)
- PRIMES (M. Agrawal, N. Kayal, and N. Saxena, 2002)
• This problem should be pretty familiar to you

**REACHABILITY**
Given: a directed graph $G$ and vertices $u, v$ of $G$
Return: $Y$ if $v$ is reachable from $u$ in $G$; $N$ otherwise.

• Obviously, REACHABILITY is in $\text{PTime}$. But what about the space-bound?
Theorem

*REACHABILITY is in NLogSpace.*

Proof.
Here is an ‘algorithm’:

```
begin reachND(G, u, v)
    n := number of vertices in G
    w := u
    c := 0
    while w ≠ v and c < n
        pick v such that w = v or there is an edge from w to v
        w := v
        increment c;
    if c < n
        return Y
    return N
end reachND
```
Exercise: try writing out definitions of the above problems in the standard form:

PROBLEM NAME
Given: ... ;
Return: ... 

We will encounter many examples of problems in other complexity classes in the coming lectures.
• Complexity classes fit inside one another in some obvious ways:

\[
\begin{align*}
\text{TIME}(G) & \subseteq \text{NTIME}(G) & \text{SPACE}(G) & \subseteq \text{NSPACE}(G) \\
\text{TIME}(G) & \subseteq \text{SPACE}(G) & \text{NTIME}(G) & \subseteq \text{NSPACE}(G)
\end{align*}
\]

• Also, if \( G \subseteq H \), then \( \text{TIME}(G) \subseteq \text{TIME}(H) \), and similarly for \( \text{NTIME}, \text{SPACE} \) and \( \text{NSPACE} \).

• Some slightly less obvious inclusions can be established using the notion of a configuration graph for a TM \( M \) on input \( x \).
• A configuration is possible state of \( M \), describing the contents of the tapes, the head positions and the state.

• A tape with symbols \( a_1, \ldots a_p \) with the head at position \( i \) (\( 1 \leq i \leq p \)) and \( M \) in state \( s \) can be conveniently encoded as a string

\[
\sigma = a_1 \cdots a_{i-1}sa_i \cdots a_p.
\]

Hence a configuration of a \( k \)-tape TM can be conveniently described by \( k \) such strings (with separators):

\[
c = \sigma_1 ; \sigma_2 ; \cdots ; \sigma_k.
\]

• If there is a bound on the space used—say \( s(n) \), where \( n \) is the length of the input \( x \), then there are at most \( 2^{O(s(n))} \) configurations to be considered.

• The set of these configurations, say \( V \), forms a graph \( G = (V, E) \) where \((c, d) \in E\) just in case \( M \) has a transition taking \( c \) to \( d \).

• We can identify a start configuration \( c_0 \) (with input \( x \)) and a success configuration \( c_* \).
Theorem
\( \text{NSPACE}(g) \subseteq \text{TIME}(2^{O(g)}) \)

Proof.
Let \( M \) be in \( \text{NSPACE}(g) \). We must show it is in \( \text{TIME}(2^{O(g)}) \).

Take any input \( x \) with \( |x| = n \). We can easily (write a deterministic TM to) construct the configuration graph, of \( M \) with space bound \( n \). But \( |G| < 2^{O(g)} \), and we may search for a path from the start configuration (with input \( x \)) to the(!) success configuration in time linear in the size of \( G \).

Corollary
\( \text{NLogSpace} \subseteq \text{PTime}, \text{NPSpace} \subseteq \text{ExpTime}, \text{NExpSpace} \subseteq 2^{-\text{ExpTime}}, \text{etc.} \)
• We observed above that $\text{TIME}(g) \subseteq \text{SPACE}(g)$ and $\text{NTIME}(g) \subseteq \text{NSPACE}(g)$.

• These inclusions can be strengthened with a simple trick.

• If $M$ is a (non-deterministic) TM running time $g$, then it makes, on input $x$ with $|x| = n$, a series of at most $g(n)$ choices as to which transition to take; each of these choices is made from a fixed list of (say) $q$ choices.

• We can represent any run as a sequence of symbols $k_1, \ldots, k_{g(n)}$ (chosen from a fixed alphabet of size $k$).
Theorem
\( \text{NTime}(g) \subseteq \text{Space}(g) \).

Sketch proof.
Let \( M \) be in \( \text{NTime}(g) \). We must show it is in \( \text{Space}(g) \).

We can write a deterministic TM \( M^* \) running in space \( g \) (just as \( M \) does) but using an extra work-tape to record the non-deterministic choices in a run of \( M \), expressed as a string \( k_1, \ldots, k_g(n) \). With only marginal space overhead, \( M^* \) can easily be made to cycle through all possible sequences \( k_1, \ldots, k_g(n) \), terminating if \( M \) would have.

Corollary
\[ \text{LogSpace} \subseteq \text{NLogSpace} \subseteq \text{PTime} \subseteq \text{NPTime} \subseteq \text{PSPACE} \subseteq \text{NPSPACE} \subseteq \text{ExpTime} \subseteq \text{NExpTime} \subseteq \text{EXPSPACE} \subseteq \text{NEXPSPACE} \subseteq 2^{-\text{ExpTime}} \cdots \]
Theorem
\( \text{NTIME}(g) \subseteq \text{SPACE}(g) \).

Sketch proof.
Let \( M \) be in \( \text{NTIME}(g) \). We must show it is in \( \text{SPACE}(g) \).

We can write a deterministic TM \( M^* \) running in space \( g \) (just as \( M \) does) but using an extra work-tape to record the non-deterministic choices in a run of \( M \), expressed as a string \( k_1, \ldots, k_{g(n)} \). With only marginal space overhead, \( M^* \) can easily be made to cycle through all possible sequences \( k_1, \ldots, k_{g(n)} \), terminating if \( M \) would have.

Corollary
\[ \text{LogSPACE} \subseteq \text{NLogSPACE} \subseteq \text{PSPACE} \subseteq \text{NPSPACE} \subseteq \text{ExpSPACE} \subseteq \text{NExpSPACE} \subseteq 2^{\text{-ExpTime}} \cdots \]
Outline

Turing machines

Time and space

A separation result

Summary
Definition
Let \( f : \mathbb{N} \to \mathbb{N} \) be ‘proper’. (There exists a TM, which given \( n \), computes \( f(n) \) symbols \( \star \) on its output tape.) The \( f \)-bounded Halting problem is the following problem:

\[
\text{HALTING}_f
\]
Given: the code, \( m \), of a deterministic Turing Machine, \( M \), and a string, \( x \), in the alphabet of \( M \);
Return: Yes if \( M \) terminates on input \( x \) in time at most \( f(|x|) \), and No otherwise.

Theorem
\( \text{HALTING}_f \notin \text{TIME}(f(\lfloor n/2 \rfloor)) \).
Proof.
Suppose HALTING$_f$ is recognized by a Turing machine $H_f$, guaranteed to terminate in time $f(\lfloor n/2 \rfloor)$. Consider the Turing machine, say $H_f^*$, with code $h_f^*$:

What happens if $H_f^*$ is given as input $h_f^*$—i.e. a description of itself as input? The embedded $H$ receives input $h_f^*$; $h_f^*$, and terminates (if at all) in time $f(|h_f^*|)$. Hence:

$$H_f^* \downarrow h_f^* \Rightarrow H_f^* \uparrow h_f^*$$
$$H_f^* \uparrow h_f^* \Rightarrow H_f^* \downarrow h_f^*$$
• However, if \( f(n) \) is a ‘proper’ complexity function, we can decide the problem \( H_f \) in time \( (f(n))^3 \) using a version, \( U_f \) of the universal Turing machine, \( U \).

• The machine \( U_f \) works as follows given input \( (M, x) \).
  - writes \( f(|x|) \) symbols \( \star \) on an ‘alarm-clock’ (work)tape;
  - simulate the steps of \( M \) in the usual way, advancing a counter on the alarm clock tape by 1 for each step;
  - abandon the computation if the alarm clock rings, and just output No.

• This machine can be made to run in time \( O(f(n)^3) \), and so can be sped-up to run in time \( f(n)^3 \).
Define $f'(n) = f(\lfloor n/2 \rfloor)$. Now, $M_f$ decides $\text{HALTING}_f$, and runs in time $f(n^3) = f'(2n+1)^3$. On the other hand, $\text{HALTING}_f$, is not computable in time $f(\lfloor n/2 \rfloor) = f'(n)$. Moreover, if $f'$ is ‘proper’, so is $f$. Hence:

**Theorem**

*For all ‘proper’ functions $f$, $\text{TIME}(f(n)) \subsetneq \text{TIME}((f(2n + 1)^3))$.*

Using similar reasoning:

**Theorem**

*For all ‘proper’ functions $f$, $\text{SPACE}(f(n)) \subsetneq \text{SPACE}(f(n) \log f(n))$.***
This yields a very important corollary:

**Theorem**

\[ \text{PTime} \subsetneq \text{ExpTime}. \]

**Proof.**

Since any polynomial is dominated by \(2^n\),

\[
\begin{align*}
\text{PTime} & \subseteq \text{Time}(2^n) \\
& \subsetneq \text{Time}(2^{3(2^n+1)}) \\
& \subseteq \text{ExpTime}.
\end{align*}
\]

Similarly

**Theorem**

\[ \text{NPTime} \subsetneq \text{NExpTime}, \text{ and } \text{PSpace} \subsetneq \text{ExpSpace}. \]
• Going back to our earlier result that

\[
\text{PTime} \subseteq \text{NPTime} \subseteq \text{PSpace} \subseteq \text{ExpTime}
\]

we know that at least one of these inclusions is strict.

• If you could say which, a great many people would like to know, especially if it turned out to be \(\text{PTime} \subset \text{NPTime}\).

• It is suspected that all of these inclusions are strict, but no one really has any idea.
• Notice the asymmetry involved in the notion of (non-deterministic) computation:

\[ M \text{ recognizes } L \subseteq \Sigma^*, \text{ just in case, for each string } x \in \Sigma^*, \]
\[ x \in L \text{ if and only if there exists a terminating run of } M \text{ on input } x. \]

• This asymmetry prompts us to define the complement classes as follows.

\[ \text{If } C \text{ is a class of languages, then } Co-C \text{ is the class of languages } L \text{ such that } \Sigma^* \setminus L \text{ is in } C, \text{ where } \Sigma \text{ is the alphabet of } L. \]
• Trivially,

\[ \text{TIME}(G) = \text{CO-TIME}(G) \]
\[ \text{SPACE}(G) = \text{CO-SPACE}(G). \]

• For non-deterministic classes, some of these equations are not known to hold:

\[ \text{NPTIME} \ ? = \text{CO-NPTIME} \]

• But there are some surprises to come . . .
Outline

Turing machines

Time and space

A separation result

Summary
Summary

• In this lecture we have
  • defined multi-tape Turing machines
  • defined the basic complexity classes based on the Turing model of computation;
  • presented some simple inclusions involving these complexity classes;
  • shown that some of these inclusions are strict.

• Reading for this lecture:
  • Sipser Ch. 3 (TMs)
  • Sipser Ch. 4 (Halting problem)
  • Sipser Ch. 7.1 (Time complexity)
  • Sipser Ch. 9.1 (Hierarchy theorem)