COMP36111: Advanced Algorithms I

Lecture 6: Problems, algorithms and complexity measures

Ian Pratt-Hartmann

Room KB2.38: email: ipratt@cs.man.ac.uk

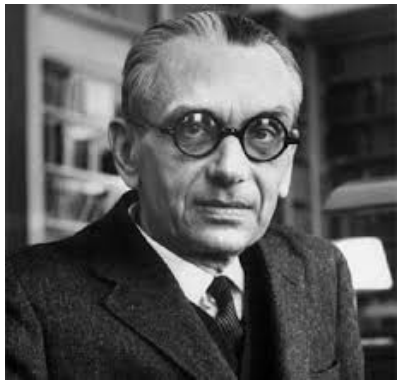
2016–17

- 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)

Outline

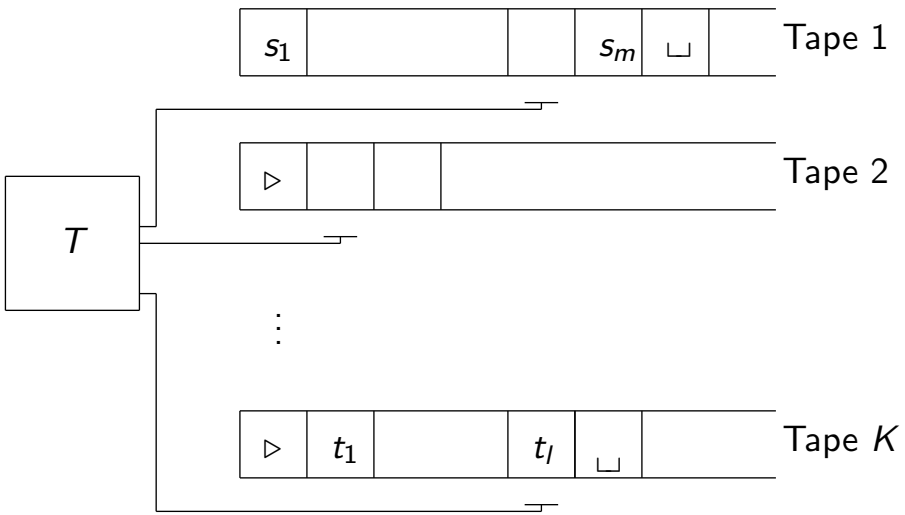
Alan Turing

Alonzo Church



Kurt Gödel

- 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*)
  - Σ 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, Σ and Q)—see below.

- A *symbol* is an elment of $\Sigma \cup \{\sqcup, \triangleright\}$
  - We pronounce $\sqcup$ as *blank* and $\triangleright$ as *start*.

- A *transition* (for K, Σ 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 $\{\texttt{left}, \texttt{right}, \texttt{stay}\}$

- 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 $\langle p, \bar{s}, q, \bar{t}, \bar{d} \rangle$.

- A configuration of M: a K-tuple of strings

$$\triangleright, s_{k,1}, \ldots, s_{k,i-1}, q, s_{k,i}, \ldots, s_{k,n(k)}.$$

  representing the situation in which the kth 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 Σ, 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 $f_M : \Sigma^* \to \Sigma^*$ as follows.

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

We say that M *computes* the function f_M. A partial function $f : \Sigma^* \to \Sigma^*$ is *computable*) if it is computed by some (deterministic) Turing machine.

- An important feature of Turing machines is that they they are finite objects of the form $\langle K, \Sigma, Q, q_0, T \rangle$.
- As such, they can be encoded (in some standard way) in any alphabet Σ. used as input to other Turing machines.
- In particular, there exists a universal Turing machine:

Theorem

Fix some alphabet Σ. There exists a Turing machine U with the following property. For any Turing Machine M with alphabet Σ, 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 Σ, and let $x \in \Sigma^*$. Assume WLOG Σ contains the symbol Yes. We say M *accepts* x if M has a halting run on input x, with output Yes.

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 *recognizable* if there is a (deterministic) Turing Machine which recognizes it.
- A language is *computable* if there is a deterministic Turing Machine M recognizing it, such that M always halts.
- Sometimes, we use the vocabulary of *problems* and *decidability*
  - language $\Longleftrightarrow$ problem
  - computable $\Longleftrightarrow$ decidable
- Sometimes, we use the following older terminology:
  - recognizable $\Longleftrightarrow$ recursively enumerable (r.e.)
  - computable $\Longleftrightarrow$ recursive

- It is easy to see from the above that a problem $\mathcal{P} \subseteq \Sigma^*$ is decidable if and only if the total function

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

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

  is computable.

- 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:

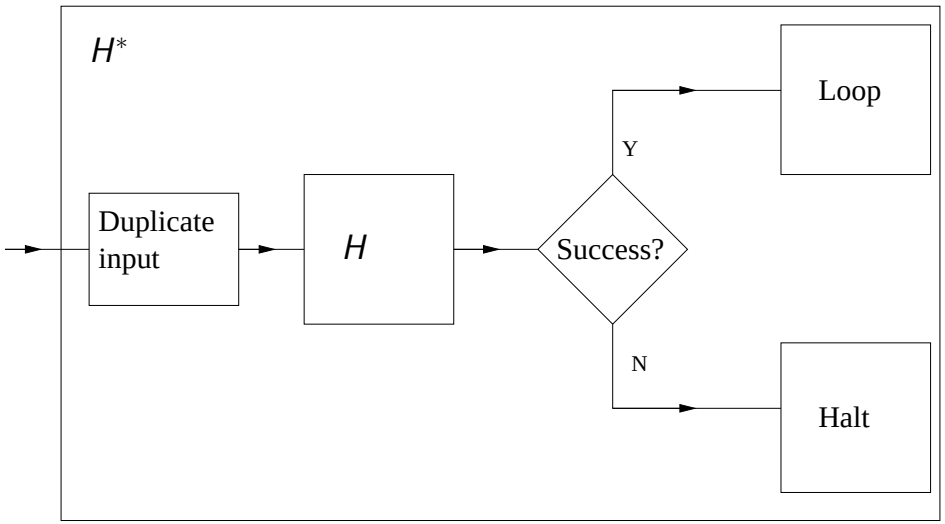
HALTING
Given: a Turing Machine description, M and a string, x in the alphabet of M,
Return: Yes if $M \downarrow x$, and No otherwise.

Theorem (Turing, 1936)

The Halting problem is not decidable.

Proof.

Suppose such a Turing machine exists, say H. Consider the Turing machine, say H^*:



What happens if H^* is given as input H^*—i.e. a description of itself as input? The embedded H receives input (H^*, H^*). Hence:

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

Outline

- 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 Σ, 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, 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 $3n^3 - 13n + 42$.
- The problem is: this sort of complexity-measure is not very *robust*:
- 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".)

Definition

Let M be a Turing machine, and G a set of functions from $\mathbb{N}$ to $\mathbb{N}$. We say that *M runs in time G* if, for some $g \in G$, M runs in time g. Similarly, we say that *M runs in space G* if, for some $g \in G$, M runs in space g.

Definition

Let $g : \mathbb{N}^k \to \mathbb{N}$ be a function. Denote by $O(g)$ the set of functions

$$O(g) = \{g' : \mathbb{N}^k \to \mathbb{N} \mid \text{ there exist } c \in \mathbb{N},\ n'_1, \ldots, n'_k \in \mathbb{N} \text{ s.t. for all } n_1 > n'_1 \ldots \text{for all } n_k > n'_k,\ g'(n_1, \ldots, n_k) \leq cg(n_1, \ldots, n_k)\}.$$

- Thus, it makes sense to say, for example, that a given Turing machine runs in time (or space) $O(n^2)$, or $O(n^3)$, or $O(2^n)$.
- This reflects the fact that additive and multiplicative constants are irrelevant: we can always speed up Turing machines arbitrarily.
- It also enables us to abstract away from irritating details of TMs
- Thus, we can say, for example, that the algorithm for MATCHING described in Lecture 0 runs in $O(n^3)$ time, without having to worry about how, exactly, all the data-structures are encoded.

Outline

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* $\textsc{Time}(G)$ (or $\textsc{Space}(G)$) if there exists a deterministic Turing machine M recognizing L, such that M runs in time (respectively, space) G.

If $G = \{g\}$, we write $\textsc{Time}(g)$ instead of $\textsc{Time}(\{g\})$; similarly for $\textsc{Space}$.

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* $\mathrm{NTime}(G)$ (or $\mathrm{NSpace}(G)$) if there exists a Turing machine M recognizing L, such that M runs in time (respectively, space) G.

If $G = \{g\}$, we write $\mathrm{NTime}(g)$ instead of $\mathrm{NTime}(\{g\})$.

- Thus, we showed in Lectures 2 and 3 that
  - REACHABILITY is in $\textsc{Time}(n)$;
  - MATCHING is in $\textsc{Time}(n^3)$.
- To think about: why did we not say:
  - REACHABILITY is in $\textsc{Time}(O(n))$;
  - MATCHING is in $\textsc{Time}(O(n^3))$?

- When talking about the complexity of problems, we typically consider *larger* classes of functions than those of the form $O(f)$.
- Here are some such classes:

$$\begin{aligned} P &= \{n^c \mid c > 0\} \\ E &= \{2^{n^c} \mid c > 0\} \\ E_2 &= \{2^{2^{n^c}} \mid c > 0\} \\ E_k &= \{2^{2^{\cdots^{2}}\}^{n^c}}_{k \text{ times}} \mid c > 0\} \end{aligned}$$

- A function $g : \mathbb{N} \to \mathbb{N}$ which is in E_k for some k is said to be *elementary*.
- It is easy to define a computable function which is non-elementary:

$$f(n) = 2^{2^{\cdots^{2}}\}_{n \text{ times}}} .$$

It grows rapidly.

Outline

- Thus we have the following deterministic complexity classes

$$\begin{aligned} \textsc{LogSpace} &= \textsc{Space}(\log n) \\ \textsc{PTime} &= \textsc{Time}(P) & \textsc{PSpace} &= \textsc{Space}(P) \\ \textsc{ExpTime} &= \textsc{Time}(E) & \textsc{ExpSpace} &= \textsc{Space}(E) \\ k\text{-}\textsc{ExpTime} &= \textsc{Time}(E_k) & k\text{-}\textsc{ExpSpace} &= \textsc{Space}(E_k). \end{aligned}$$

- And likewise their non-deterministic counterparts:

$$\begin{aligned}
\text{NPTime} &= \text{NTime}(P) \\
\text{NExpTime} &= \text{NTime}(E) \\
\text{N}k\text{-ExpTime} &= \text{NTime}(E_k)
\end{aligned}$$

$$\begin{aligned}
\text{NLogSpace} &= \text{NSpace}(\log n) \\
\text{NPSpace} &= \text{NSpace}(P) \\
\text{NExpSpace} &= \text{NSpace}(E) \\
\text{N}k\text{-ExpSpace} &= \text{NSpace}(E_k).
\end{aligned}$$

- Thus, we showed in earlier lectures that
  - DIRECTED GRAPH CYCLICITY
  - GRAPH CONNECTEDNESS
  - PERFECT MATCHING
  - STRING OCCURRENCE
  - LINEAR PROGRAMMING FEASIBILITY (well, we stated this)

  are all in PTIME
- 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 obvious ways:

$$\textsc{PTime} \subseteq \textsc{ExpTime} \subseteq \text{2-}\textsc{ExpTime} \subseteq \cdots$$
$$\textsc{LogSpace} \subseteq \textsc{PSpace} \subseteq \textsc{ExpSpace} \subseteq \text{2-}\textsc{ExpSpace} \subseteq \cdots$$
$$\textsc{NPTime} \subseteq \textsc{NExpTime} \subseteq \text{2-}\textsc{NExpTime} \subseteq \cdots$$
$$\textsc{NLogSpace} \subseteq \textsc{NPSpace} \subseteq \textsc{NExpSpace} \subseteq \text{2-}\textsc{NExpSpace} \subseteq \cdots$$

- Actually, the non-deterministic classes can be interleaved:

$$\textsc{PTime} \subseteq \textsc{NPTime} \subseteq \textsc{ExpTime} \subseteq \textsc{NExpTime} \subseteq \cdots$$

- There are some non-obvious inclusions, which we will discuss later.
- Most obvious problem: is any of these inclusions strict?

Definition

The f-bounded *Halting problem* is the following problem:

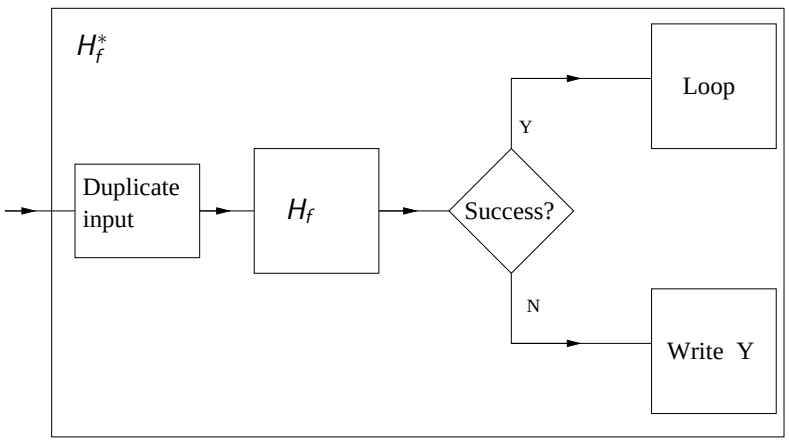
$\underline{\text{HALTING}}_f$
Given: a Turing Machine description, M and a string, x in the alphabet of M,
Return: Yes if $M \downarrow x$ in time $\leq f(|x|)$, and No otherwise.

Theorem

$HALTING_f \notin \textsc{Time}(f(\lfloor n/2 \rfloor))$.

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^*:



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 HALTING$_f$, and runs in time $f(n^3) = f'(2n+1)^3$. On the other hand, 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, $\textsc{Time}(f(n)) \subsetneq \textsc{Time}((f(2n+1)^3)$.

Using similar reasoning:

Theorem

For all 'proper' functions f, $\textsc{Space}(f(n)) \subsetneq \textsc{Space}(f(n) \log f(n))$.

This yields a very important corollary:

Theorem

$\textsc{PTime} \subsetneq \textsc{ExpTime}$.

Proof.

Since any polynomial is dominated by 2^n,

$$\begin{aligned} \textsc{PTime} &\subseteq \textsc{Time}(2^n) \\ &\subsetneq \textsc{Time}(2^{3(2n+1)}) \\ &\subseteq \textsc{ExpTime}. \end{aligned}$$

□

Similarly

Theorem

$\textsc{NPTime} \subsetneq \textsc{NExpTime}$, *and* $\textsc{PSpace} \subsetneq \textsc{ExpSpace}$.

- Notice the asymmetry involved in the notion of (non-deterministic) computation:

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

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

  If $\mathcal{C}$ is a class of languages, then Co-$\mathcal{C}$ is the class of languages L such that $\Sigma^* \setminus L$ is in $\mathcal{C}$, where Σ is the alphabet of L.

- Trivially,

$$\begin{aligned} \textsc{Time}(G) &= \textsc{Co-Time}(G) \\ \textsc{Space}(G) &= \textsc{Co-Space}(G). \end{aligned}$$

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

$$\textsc{NPTime} \quad ? = \quad \textsc{Co-NPTime}$$

- But there are some surprises to come . . .