16.1 Countability

Proving the non-existence of algorithms for computational problems can be very difficult. Indeed, we do not know how to prove \( P \neq \text{NP} \). So a natural question is whether can show that there are any problems outside of \( P \). Here we will show that in fact that “most” problems are outside \( P \). In fact, most problems have no algorithm whatsoever (regardless of efficiency). We will show this by a counting argument, arguing that there are (many) “more” languages than there are algorithms. To do this, we need be able to distinguish the sizes of infinite sets (as there are infinitely many languages and infinitely many algorithms).

Definition 16.1 A set \( S \) is countably infinite iff there is a bijection \( f : \mathbb{N} \rightarrow S \).\(^1\) \( S \) is countable iff it is either finite or countably infinite. A set \( S \) is uncountable if it is not countable.

Countably infinite sets are the “smallest” infinite sets. (Every infinite set contains a countably infinite set.)

Proposition 16.2 A set is countable iff there is a surjection \( f : \mathbb{N} \rightarrow S \).\(^2\)

That is, a set \( S \) is countable iff we can “enumerate” the elements of \( S \) in a sequence \( f(0), f(1), f(2), \ldots \) (which may have repetitions). The proof of this proposition is a good exercise.

Examples of countable sets:

- \( \mathbb{N} \)
- \( \mathbb{Z} \)
- \( \mathbb{N} \times \mathbb{N} \)
- \( \mathbb{Q} \)
- \( \Sigma^{*} \) for any finite, nonempty alphabet \( \Sigma \).

---

\(^1\)A bijection is a function that is one-to-one and onto.
\(^2\)A surjection is an onto function.
• The class $R$ of recursive (=decidable) languages over an alphabet $\Sigma$. (Recall that a language $L$ is decidable iff there is an algorithm $M$ such that for every input $x$, $M(x)$ halts and outputs 1 or 0 according to whether or not $x \in L$.)

• The class $R.E.$ of recursively enumerable (=recognizable) languages over an alphabet $\Sigma$. (Recall that a language $L$ is recognizable iff there is an algorithm $M$ such that $L = L(M) \overset{\text{def}}{=} \{x : M(x) \text{ accepts } x\}$.

We record some useful properties of countable sets (proving them is a good exercise).

**Theorem 16.3**  
1. Every subset of a countable set is countable.

2. The union of countably many countable sets is countable.

3. The direct product of two countable sets is countable.

4. The image of a countable set under any function is countable.

Now, we need some uncountable sets.

**Theorem 16.4** If $S$ is infinite, then $P(S)$ is uncountable.

**Proof:**

• Suffices to prove for $S = \mathbb{N}$ (why?).

• Suppose for contradiction that $P(\mathbb{N})$ were countable.

• Then there is an enumeration of all subsets of $\mathbb{N}$ say $P(\mathbb{N}) = \{S_0, S_1, \ldots\}$

**Diagonalization**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_i$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S_0$</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>$S_1$</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>$S_2$</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>$S_3$</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

"Y" in row $i$, column $j$ means $j \in S_i$
• Let $D = \{ i \in \mathbb{N} : i \in S_i \}$ be the diagonal.

\[ D = YNY \ldots = \{0,3,\ldots\} \]

• Let $\overline{D} = \mathbb{N} - D$ be its complement.

\[ \overline{D} = NYY \ldots = \{1,2,\ldots\} \]

• **Claim:** $\overline{D}$ is omitted from the enumeration, contradicting the assumption that every set of natural numbers is one of the $S_i$s.

**Pf:** $\overline{D}$ is different from each row because they differ at the diagonal.

Corollary 16.5 For every nonempty finite alphabet $\Sigma$, there are uncountably many languages over $\Sigma$. In particular, there are uncountably many non-r.e. languages.

### 16.2 Universal Algorithms

The argument in the previous section is nonconstructive — it tells us that there many undecidable languages, but doesn’t immediately tell us how to get our hands on any of them. In the next section, we will see how to make it more constructive and deduce that specific problems are undecidable. These will initially be problems where the input itself includes the description of an algorithm (much like the languages like ALL$_{\text{DFA}}$ you considered on HW8, where the input is a finite automaton). So first we should discuss what can be done for such problems.

**Theorem 16.6 (Universal Word RAM)** There is a single Word-RAM program $U_{\text{WR}}$ such that for every Word-RAM program $P$ and input $x \in \mathbb{N}^*$ and $k \geq \max_i x_i$, $U_{\text{WR}}(\langle P, x, k \rangle)$ has the same behavior as $P(x; k)$. [Here $k$ represents the bound on the size of numbers given to $P$.]

That is, if $P(x;k)$ halts with an output of $y$, so does $U_{\text{WR}}(\langle P, x, k \rangle)$. If $P$ does not halt, neither does $U_{\text{WR}}$. Moreover, if $P(x;k)$ halts in $T_P(x;k)$ steps, then $U_{\text{WR}}(\langle P, x, k \rangle)$ halts in $O(T_P(x;k) + O(|\langle P, x, k \rangle|))$.

**Proof:** [sketch] Registers of $U_{\text{WR}}$:

- $q_P =$ number of lines in $P$
- $r_P =$ number of registers used by $P$
- program counter $\ell_P =$ current line number in simulation of $P$
- $2^w_P$ where $w_P$ is current word size of $P$
- $S_P$ current space usage of $P$
- Scratch registers

Memory of $U_{\text{WR}}$:
• first $4q_p$ locations = lines of the program $P$, with each line being described by 4 values:
  - Instruction type (0–11, indicating +, -, ·, MALLOC, HALT, etc.)
  - register numbers $i, j, k$, constants $m$, or line number $\ell$ used in instruction
• next $r_p$ locations = registers of $P$
• then we put the memory of $P$ (one location of $U_{WR}$’s memory per location of $P$’s memory)

Then each step of $P$ can be simulated using $O(1)$ steps of $U_{WR}$. Need $O(|\langle P, x, k \rangle|)$ steps at beginning to format initial configuration and $O(T_P(x, k))$ steps at end to format output configuration.

**Corollary 16.7 (Universal TM)** There is a single (multitape) Turing Machine $U_{TM}$ such that for every (multitape) Turing Machine $M$ and input $x \in \Sigma^*$, $U(\langle M, x \rangle)$ has the same behavior as $M(x)$.

Moreover, if $M(x)$ halts in $T_M(x)$ steps, then $U_{TM}(\langle M, x \rangle)$ halts in $\tilde{O}(T_M(x) + |\langle M, x \rangle|)^2$ steps, where $\tilde{O}(f)$ is shorthand for $f \cdot \text{poly} (\log f)$.

**Proof:** We wish to construct a $U_{TM}$ which simulates some input Turing machine $M$ on some input string $x$. Although not shown in class, it is possible to construct a word RAM program $P$ which has the same behavior as $M$ on any input, with only $O(1)$ factor slowdown in running time. Then $P$ can be simulated by $U_{WR}$, which in turn by Theorem 7.2 of Lecture 7 can be simulated by a multitape TM with slowdown as in the current corollary statement.

The Universal TM (introduced by Turing) had important technological significance. It meant that computers could be programmable. Instead of building special-purpose hardware for each algorithm you want to implement (what people had done before — essentially constructing calculators), you could build one general-purpose machine that can implement any desired algorithm. Universality also has significance for software. For example, each new generation of C compilers is written using the previous generation of C compilers!

We are interested in the following theoretical implication of the Universal Machines:

**Corollary 16.8** The ACCEPTANCE PROBLEMS for Turing machines and Words-RAMs are recognizable (a.k.a. recursively enumerable):

1. $A_{TM} = \{ \langle M, x \rangle : M \text{ at TM that accepts } x \}$.
2. $A_{WR} = \{ \langle P, x, k \rangle : P \text{ a Word-RAM such that } P(x, k) \text{ accepts} \}$.

A natural question is whether these languages are decidable (a.k.a. recursive).

### 16.3 Undecidability of the Acceptance Problem

$A_{TM}$ is a natural choice for showing that undecidability because it is r.e.-complete. That is, every r.e. language $L = L(M)$ reduces to $A_{TM}$:

Before proving its undecidability, it’s useful to make a simplifying detail — that every string represents a TM.
• Let $\Sigma$ be the alphabet over which TMs are represented (that is, $\langle M \rangle \in \Sigma^*$ for any TM $M$)

• Let $w \in \Sigma^*$

• if $w = \langle M \rangle$ for some TM $M$ then $w$ represents $M$

• Otherwise $w$ represents some fixed TM $M_0$ (say the simplest possible TM).

**Theorem 16.9** $A_{TM}$ is not recursive.

**Proof:**

• Look at $A_{TM}$ as a table answering every question:

<table>
<thead>
<tr>
<th>$M_0$</th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>$Y$</td>
<td>$N$</td>
<td>$N$</td>
<td>$Y$</td>
</tr>
<tr>
<td>$M_1$</td>
<td>$Y$</td>
<td>$Y$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>$M_2$</td>
<td>$N$</td>
<td>$N$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>$M_3$</td>
<td>$Y$</td>
<td>$Y$</td>
<td>$Y$</td>
<td>$Y$</td>
</tr>
</tbody>
</table>

$M_0$ accepts $w_j$ iff $M_j$ accepts $w_j$

• If $A_{TM}$ were recursive, then so would be the diagonal $D$ and its complement.

  - $D = \{w_i : M_i$ accepts $w_i\}$.
  - $\bar{D} = \{w_i : M_i$ does not accept $w_i\}$.

• But $\bar{D}$ differs from every row, i.e. it differs from every r.e. language. $\Rightarrow \Leftarrow$.

---

**Unfolding the Diagonalization**

• Suppose for contradiction that $A_{TM}$ were recursive.

• Then there is a TM $M^*$ that decides $\bar{D} = \{\langle M \rangle : M$ does not accept $\langle M \rangle\}$:

• Run $M^*$ on its own description $\langle M^* \rangle$.

• Does it accept?

  $M^*$ accepts $\langle M^* \rangle$

  $\iff \langle M^* \rangle \in \bar{D}$

  $\iff M^*$ does not accept $\langle M^* \rangle$. 
• Contradiction!

**Corollary 16.10** $A_{WR}$ is undecidable. (And same for the Acceptance Problem in any universal model of computation.)

Alan Mathison Turing (1912-1954)

24 years old when he published *On computable numbers* …

### 16.4 Non-Recognizable Languages

**Theorem 16.11** A language $L$ is decidable iff $L$ is recognizable and co-recognizable. That is, $R = \text{R.E.} \cap \text{co-R.E.}$.

**Proof:** Suppose $L$ is decidable. Then $M$ recognizes $L$, and the Turing machine which halts with the opposite output of $M$ recognizes $\bar{L}$. Thus $L$ is both recognizable and co-recognizable.

Now suppose $L$ is both recognizable, via some TM $M_1$ and co-recognizable, via some TM $M'$. That is, if $x \in L$ then $M$ halts on $x$ in an accept state and $M'$ either rejects or runs forever, and if $x \notin L$, then $M'$ halts on $x$ in an accept state and $M$ either reject or runs forever. To decide $L$, we run $M$ and $M'$ on $x$ in parallel. That is, using multiple tapes, we use one tape to simulate $M$ on $x$ and another to simulate $M'$ on $x$. We alternate taking one step of $M$, followed by one of $M'$, then one of $M$, etc. Eventually one of $M$ or $M'$ must halt (since either $x \in L$ or $x \notin L$), so as soon as either one halts, we can determine whether $x \in L$ or $x \notin L$. 

**Corollary 16.12** $\overline{A_{TM}}$ and $\overline{A_{WR}}$ are not recognizable.