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 $\neq$ 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 $f : \mathbb{N} \to S$.*[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 $f : \mathbb{N} \to S$.*[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:

- $\mathbb{N}$
- $\mathbb{Z}$
- $\mathbb{N} \times \mathbb{N}$
- $\mathbb{Q}$
- Σ^* for any finite, nonempty alphabet Σ.

[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 Σ. (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 Σ. (Recall that a language L is recognizable iff there is an algorithm M such that $L = L(M) \stackrel{\text{def}}{=} \{x : M(x) \text{ accepts } x\}$.

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.*

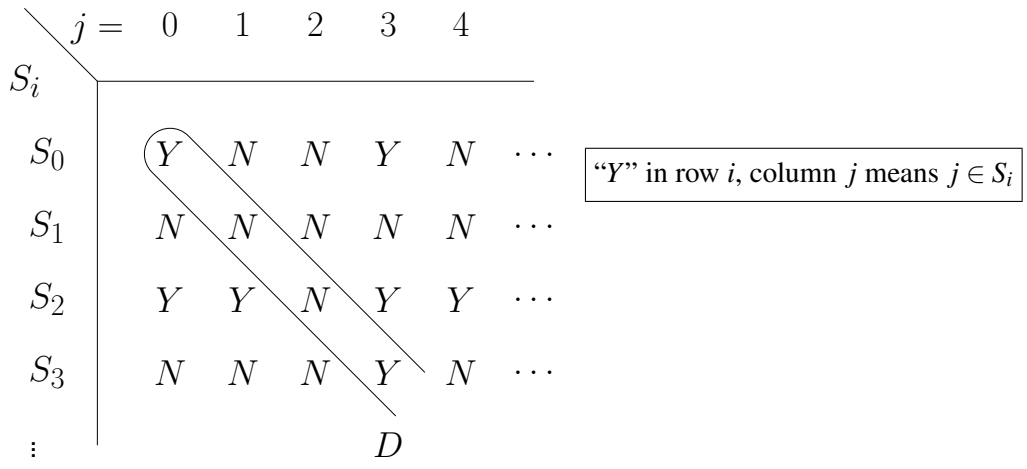
Now, we need some uncountable sets.

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 $P(\mathcal{N})$ were countable.
- Then there is an enumeration of all subsets of $\mathcal{N}$ say $P(\mathcal{N}) = \{S_0, S_1, \ldots\}$

Diagonalization



- Let $D = \{i \in \mathbb{N} : i \in S_i\}$ be the diagonal.

  $D = YNNY\ldots = \{0,3,\ldots\}$

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

  $\overline{D} = NYYN\ldots = \{1,2,\ldots\}$

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

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

■

Corollary 16.5 *For every nonempty finite alphabet Σ, there are uncountably many languages over Σ. 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_{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_{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_{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_{WR}:

- q_P = number of lines in P
- r_P = number of registers used by P
- program counter ℓ_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_{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 ℓ 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 $x \in \Sigma_M^*$, $U(\langle M,x\rangle)$ has the same behavior as $M(x)$.*

Moreover, if $M(x)$ halts in $T_M(x)$ steps, then $U_{\text{TM}}(\langle M,x\rangle)$ 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. $A_{\text{TM}} = \{\langle M,x\rangle : M \text{ at TM that accepts } x\}$.
2. $A_{\text{WR}} = \{\langle P,x,k\rangle : P \text{ a Word-RAM such that } P(x;k) \text{ accepts}\}$.

A natural question is whether these languages are decidable (a.ka. 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 Σ 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:

	w_0	w_1	w_2	w_3	
M_0	Y	N	N	Y	
M_1	Y	Y	N	N	(WLOG assume
M_2	N	N	N	N	every string w_i
M_3	Y	Y	Y	Y	encodes a TM M_i)

- Entry matching (M_i, w_j) is Y iff M_i accepts w_j
- If A_{TM} were recursive, then so would be the diagonal D and its complement.
  - $D = \{w_i : M_i \text{ accepts } w_i\}$.
  - $\overline{D} = \{w_i : M_i \text{ does not accept } w_i\}$.
- But $\overline{D}$ 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 $\overline{D} = \{\langle M \rangle : M \text{ does not accept } \langle M \rangle\}$:

- Run M^* on its own description $\langle M^* \rangle$.
- Does it accept?

M^* accepts $\langle M^* \rangle$
$\Leftrightarrow \langle M^* \rangle \in \overline{D}$
$\Leftrightarrow M^*$ 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, $R = R.E. \cap co\text{-}R.E.$.*

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 $M_¡$ 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 *$\overline{A_{\mathrm{TM}}}$ and $\overline{A_{\mathrm{WR}}}$ are not recognizable.*