Lecture IV
ELEMENTS OF COMPUTABILITY

Lecture 4: Honors Theory, Spring'02.

We have seen two language classes, *REG* and *CFL*. This section introduces two much larger classes, *REC* and *RE*. In fact, we will have $REG \subseteq CFL \subseteq REC \subseteq RE$ where all the inclusions are proper. The new classes are defined using the Turing model of computation. Basic properties of these classes are shown, via diagonalization and universality arguments.

The background of this material goes back to foundational studies in the 1930's when mathematicians began to formalize the concept of computability. Various models were studied but they all turned out to be equivalent in the sense that what is computable in one model was also computable in another, and vice-versa. The model introduced by Turing [2] is only one of them. It is chosen here because of its simplicity and resemblance to modern computers in its essence.

§1. Simple Turing Machines

We introduce a class of automata called **Simple Turing Machines** (STM). This is a bare bones version[1] of Turing's model of computation. The Turing model can be elaborated in many ways. For instance, we will later introduce a version that is suitable for complexity considerations.

A STM is a 6-tuple $M = (Q, \Sigma, \delta, q_0, q_A, \sqcup)$ where Q is a finite set of states, Σ is an alphabet, $q_0 \in Q$ is the start state, $q_A \in Q$ the accept state, $\sqcup \in \Sigma$ is the **blank symbol** and

$$\delta \subseteq Q \times \Sigma \times Q \times \Sigma \times \{-1, 0, +1\}.$$

represents the transition rules, or **instructions** (this terminology makes the connection to modern computers). This is rather similar to finite automaton except we now have an implicit **tape** with infinitely many tape **cells** indexed by the integers. The ith cell ($i \in \mathbb{Z}$) can store any symbol in Σ; thus the tape contents can be viewed as a function $T : \mathbb{Z} \to \Sigma$. The Turing machine M has a **current state** $q \in Q$ as usual, but it also has **tape head** that is positioned at some tape cell whose symbol it is **being scanned**. We say its **current (head) position** is i if it is scanning the ith cell. If $(q, a, q', a', D) \in \delta$, we will indicate this by writing the 5-tuple in the style of transition rules,

$$(q, a \to q', a', D). \tag{1}$$

This instruction is **executable** if the current state is q and head is scanning symbol a. The pair (q, a) is the "precondition" of the instruction. There may be several executable instructions (thus, M can be nondeterministic). We say M is **deterministic** in case no more than one instruction is executable; equivalently, no two instructions have the same precondition. To execute instruction (1), we enter state q', change symbol a to a' and move to position $i + D$ where i is the current head position. Thus D indicates the direction of head movement.

State Diagrams. Again, we can represent a STM using state diagrams. This is again a graph whose vertex set is Q and edges are labeled by one or more instructions. The instruction (1) will show up as the label $(a \to a', D)$ for the edge (q, q'). Consider the STM of figure 1 which recognizes the language of "duplicate words", $L_{\mathtt{dup}} := \{w\#w : w \in \{0,1\}^*\}$. The reader is asked to verify its correctness, as an exercise in familiarizing oneself with STMs.

[1] This is basically the original model described by Turing, except for the introduction of nondeterminism.

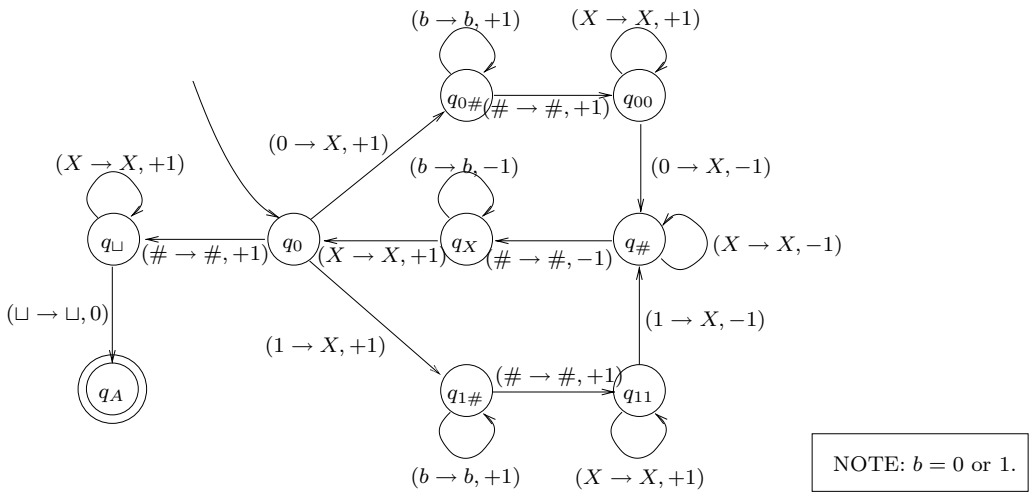

Figure 1: STM for $\{w\#w : w \in \{0,1\}^*\}$.

To understand the construction, the idea is to "cross out" the bits ($b = 0, 1$) that has been processed, replacing them by an X. Assuming the input has the form $u\#v$ ($uv \in \{0,1\}^*$), then at the beginning of the ith stage, the tape contents is $X^i u' \# X^{i-1} v'$ where u', v' are suffixes of u, v that remain to be crossed out.

Using Turing machines. We use M in one of three capacities: as an **acceptor** of a language $L(M)$, as a **transducer** to compute a partial function $t_M : \Sigma^* \to \Sigma^*$, and as a **generator** of a language $G(M)$. In each case, we need additional conventions.

- To view M as a generator of language over $\Sigma_\sqcup = \Sigma \setminus \{\sqcup\}$, we introduce a special **output state** $q_o \in Q$. The machine begins its computation on a blank tape. In the course of computation, the tape will contain a finite number of non-blank non-empty words, separated from each other by one or more blanks. Whenever M enters state q_o, we declare that the non-blank word that is being scanned on the tape is "generated". If the symbol being scanned is a blank, then the empty word is generated. The set of all generated words is denoted by $G(M) \subseteq \Sigma_\sqcup$. Note that the words in $G(M)$ may be generated more than once. Note that the state q_o is not meant to be a terminal state; it is expected to be (re)entered infinitely often, in order to generate an infinite language.
- For acceptors and transducers, we need an input convention. Input words have the form $w \in \Sigma_\sqcup^*$. We assume that $w[i]$ is placed at cell i ($i = 1, \ldots, |w|$), and everywhere else, the cells are blank. The initial head position is 1.
- Output convention. This depends on whether we view M as an acceptor of a language $L(M)$ or a transducer computing a function $t_M : \Sigma_\sqcup^* \to \Sigma_\sqcup^*$. For an acceptor, M accepts its input if some computation path enters the final state q_A. As a transducer, M on input w will output the non-blank word $t_M(w)$ that is being scanned when the machine enters the final state. (In case the scanned symbol is $\sqcup$, define $t_M(w) = \epsilon$.) Since M is nondeterministic, different paths can give different outputs. We may declare $t_M(w)$ to be undefined when this happens, or if no paths give any output.

Computation. Now that we have an informal idea of how STM computes, we may formalize the computation process. Let M be in state q, with head position $h \in \mathbb{Z}$ and $T : \mathbb{Z} \to \Sigma$ is the current tape contents. We can represent this information using the notion of a configuration. First, we must assume that the work

tape is blank everywhere except for finitely many exceptions. Choose the range $[j..k] \subseteq \mathbb{Z}$ such that (a) $T(i) = \sqcup$ for all i outside $[j..k]$, and (b) $j \leq h \leq k$. This range can be made unique by specifying it to be minimum subject to (a) and (b). Let $u \in \Sigma^*$ represent the contents of tape cells in the range $[j..k]$, *i.e.*, $u[i] = T(j+i-1)$ for $i = 0, \ldots, k-j$, Also, let $u = vw$ such that $|v| = h - j$. It follows that the currently scanned symbol in $w[1]$. Note that this requires $|w| \geq 1$. The current configuration is then represented by (v, q, w) or, simply, vqw. Thus, the set of configurations can be taken to be

$$\Sigma^* \times Q \times \Sigma^+$$

If C, C' are configurations, we define $C \vdash_M C'$ if the configuration C can move to the configuration C' by executing an instruction of M. Suppose the instruction being executed is (1), and $C = vqw$ and $C' = v'q'w'$. There are three possibilities for the direction D:

CASE $D = 0$**:** Then $v' = v$ and w' is the same as w except that $w'[1] = a'$.

CASE $D = +1$**:** Then $v' = va'$. Let $w = aw_1$. In case $w_1 = \epsilon$, then $w' = \sqcup$. Otherwise, $w' = w_1$.

CASE $D = -1$**:** Assume $w = aw_1$. In case $v = \epsilon$, we have $v' = \epsilon$ and $w' = \sqcup a' w_1$. Otherwise, let $v = v_1 b$ for some symbol b. Then $v' = v_1$ and $w' = ba'w_1$.

The reflexive, transitive closure of $\vdash_M$ is denoted $\vdash_M^*$. We drop the subscript "M" when it is understood. The **initial configuration** on input w is simply $q_0 w$, denoted $C_0(w)$ (note that it does not depend on M). An **accepting configuration** is one with state q_A. We say M **accepts** w if $C_0(w) \vdash_M^* C$ for some accepting C. $L(M)$ is the set of words accepted by M.

Infinite Computations. The preceding definitions is a minimal set for defining STM computation. But we must refine our notations to discuss what happens when a word is not accepted by C. In this case, there are two distinct behaviors we want to capture. We call configuration C **terminal**. if there does not exist C' such that $C \vdash C'$. We may assume that the accepting configuration is always terminal. A terminal but non-accepting configuration has no executable instruction; it is said to be **stuck**. A **computation sequence** π is a finite or ω-sequence of configurations of the form

$$\pi : C_1 \vdash C_2 \vdash C_3 \vdash \cdots.$$

We call π a **computation path** if, in addition, C_1 is an initial configuration and either the sequence is infinite or it is finite with the last configuration being terminal. We also want to view the entire **computation tree** of M on input w. This tree, denoted $T_M(w)$, is rooted at $C_0(w)$ such that the set of computation paths of M on w is precisely the set of maximal paths of the tree. A basic fact about this tree is that it has bounded branching factor (this factor is at most the number of instructions in M). We say that M **halts** on w iff this tree is finite; in the contrary case, we say it **loops** on w. The notations

$$M(w)\downarrow, \quad M(w)\uparrow$$

are (respectively) used to indicate these two dispositions of M. (The fact that we introduce these special notations signals their importance.) We say M **rejects** w in case $M(w)\downarrow$ but M does not accept w. Thus, *rejection has a stronger requirement than non-accepting.* A **halting STM** M is one that halts on every input. Equivalently, for every input w, M either accepts or rejects w.

§2. Recursively Enumerable and Recursive Languages

A language is said to be **recursively enumerable** (r.e.) if it is accepted by some STM. The class of all r.e. languages is denoted RE. A language is **recursive** if it is accepted by some halting STM. The class of all recursive languages is denoted REC.

Suppose M accepts L. We also say that M **recognizes** or **semi-decides** L. If M happens to be halting, we say M **decides** L. It is common to refer to r.e. languages and recursive languages as r.e. sets or recursive sets. This arose from the context where these definitions were applied to subsets of $\mathbb{N}$ (see discussions below).

For our first result, we show that the non-determinism in STM is not essential as far as as the definitions of RE and REC were concerned.

Theorem 1 *For every STM M, there exists a STM M' such that M' is deterministic and $L(M) = L(M')$. Moreover, if M is halting, so is M'.*

Proof. We only sketch a proof: on any input w, M' will attempt to simulate every computation path of M starting on w. It uses a breadth first search of the computation tree $T_M(w)$. At any stage of the simulation, the tape of M' will hold a string of the form

$$\#C_1\#C_2\#\cdots\#C_{i-1}\$C_i\#\cdots\#C_m$$

where C_i's are configurations of M, separated by the special symbols # and \$. There is only one copy \$, and if this precedes C_i, it signifies that C_i is currently being "expanded". The configurations $C_1, \ldots, C_{i-1}$ are all at some level ℓ of the computation tree, while $C_i, \ldots, C_m$ are at level $\ell - 1$. To expand C_i, we simply replace it by all those C' such that $C_i \vdash_M C'$. To create space for this expansion, M' may have to move all the $C_{i+1}, \ldots, C_m$ to the right. Sometimes, this expansion is really a contraction (for instance, if C_i is terminal). After we expanded C_i, we will next work on C_{i+1} (indicated by moving \$ next to C_{i+1}). The simulation accepts if any of the expanded configurations is accepting. It also rejects when there are no more configurations left to be expanded (this means the computation tree is finite). Thus, if M is halting, so is M'. This concludes our sketch. **Q.E.D.**

It is clear from our definitions that

$$REC \subseteq RE.$$

A more interesting result is the following.

Corollary 2 *The recursive languages are closed under complement: $REC =$ co-REC.*

Proof. It is a simple fact that for any class K, we have $K =$ co-K iff $K \subseteq$ co-K. Hence we only have to show that if L is recursive, then co-L is recursive. Applying the previous theorem, let M be a halting deterministic STM that decides L. We construct $\overline{M}$ that acts like M, except that $\overline{M}$ accepts iff M rejects. **Q.E.D.**

Theorem 3 *The recursive languages are those r.e. languages whose complements are also r.e.. Thus, $REC = RE \cap$ co-RE.*

Proof. One direction is easy: if $L \in REC$ then the previous lemma shows that co-$L \in REC$ and hence $L \in$ co-REC and hence $L \in REC \cap$ co-$REC \subseteq RE \cap$ co-RE.

Conversely, let $L \in RE \cap$ co-RE. Then there are deterministic STM's M and M' such that $L = L(M)$ and co-$L = L(M')$. We construct a STM M'' that, on input w, simulates M on w and M' on w. The trick is that we must not naively simulation either M or M' to completion, since we cannot be sure if either will halt. But we know that for every input w, at least one of M or M' will halt: if $w \in L$ then M will halt, and if $w \notin L$ then M' will halt. Hence we simulate one step of M and one step of M' alternately. As soon as

either M or M' halts, we can accept or reject in the appropriate manner: we accept iff either M accepts or M' rejects. **Q.E.D.**

Let us prove one more result, to relate the context free languages to our new classes.

Lemma 4 *The class CFL of context free languages is a proper subset of REC.*

Proof. We sketch the argument that $CFL \subseteq REC$. Assume that P is a pda and we need to construct a halting STM M such that $L(M) = L(P)$. On any input w, M first converts the tape contents into $w\#$ where $\#$ is a special symbol. In general, the tape contents has the form $w'\#v$ where w' is a suffice of w (indicating that P is currently reading $w'[1]$ and v is the stack contents (the top of stack is the last symbol of v). Each nondeterministic step of P is easily simulated by M (which is also nondeterministic). The simulation halts when $w' = \epsilon$ – thus M is halting. M will accept iff P accepts.

To see that CFL is in fact a proper subset of REC, we note that the language $L_{\texttt{dup}}$ of duplicate words is not context free. This can be seen by a simple application of the pumping lemma for context free languages. However, the construction in figure 1 in our STM introduction shows that this language is in fact in REC. **Q.E.D.**

Although nondeterminism was not essential in our definition of RE and REC, the preceding proof illustrates its usefulness in some proofs.

Computability Dictionary. There are at least 3 independent sets of terminology in this area. The reader should feel comfortable switching among them as convenient, because they represent different computational viewpoints. The following table places the three terminology alongside each other for comparison.

	MODEL	PARTIAL COMPUTABILITY	TOTAL COMPUTABILITY
1.	Generator	recursively enumerable set (r.e.)	recursive set
2.	Acceptor	semidecidable or recognizable set	decidable set
3.	Transducer	(partial) computable function	total computable function

To further clarify this table, we delve into history. The subject of computability theory is "computational problems". The obvious form of a computational problem is the functional form. For instance, the problems of multiplication or square-root correspond to the functions $g(x, y) = xy$ and $f(x) = \sqrt{x}$, respectively. To capture the essence of computability, and to avoid considerations of precision in the representation of real numbers, computability theory is usually formulated as the study **number theoretic functions**, *i.e.*, $f : \mathbb{N}^n \to \mathbb{N}$ for some $n \geq 1$. By suitable encodings, we can even assume $n = 1$. So the square root function must be re-formulated as $f(x) = \lfloor \sqrt{x} \rfloor$. Such are the functions mentioned in the last row of the table.

We can further restrict our number theoretic functions so that range has only two values (Yes/No, 0/1, accept/reject). These are called[2] **decision problems**. Such a function $f : \mathbb{N} \to \{0, 1\}$ can be identified with the set $\{n \in \mathbb{N} : f(n) = 1\}$. Thus the characterization of computable functions is transformed into the characterization of the "decidable subsets" of $\mathbb{N}$. For any alphabet Σ, there are bijections between $\mathbb{N}$ and Σ^* and hence this is equivalent to studying decidable languages, which is our main viewpoint. The sets mentioned in rows 1 and 2 can be interpreted as subsets of $\mathbb{N}$ or languages. Note that our "official definition" of these concepts uses the terminology for generators, even though we mainly treat acceptors.

[2]This is the German word "Entscheidungsproblem" in Turing's original paper [2].

The first aim of computability theory is to characterize the "computable" (number theoretic) functions. It was quickly realized that "computable" can be refined into "partial computable" and "total computable". This accounts for the two columns (under "partial computability" and "total computability"). Likewise "decidable" can be refined into "semi-decidable" or "total decidable". Note that the presence of the qualifiers (partial, semi-, total, etc) immediately places concept in the right column. The trouble is that when the qualifier is omitted, we need some convention to tell us which column is meant.

What about other models of computation? The introduction noted that there are other models of computability. So why did we choose Turing's model? One answer is that Turing's model is remarkably flexible: when we study complexity theory later, we will see how it is easily modified to model various phenomenon in complexity. Furthermore, **Turing's Thesis** says that *anything that is algorithmically computable is computable by an algorithm is computable by Turing machines.* This is called a thesis (not a theorem) because the concept of "algorithmically computable" is informal. But given any reasonable interpretation, we will find that the thesis holds up. The converse of this thesis is clear: whatever an "algorithmically computable" means, the method used by a STM to accept a language or to compute a function qualifies as algorithmic. This justifies equating the classes RE and REC with the concepts of semi-decidable and total decidable sets.

§3. Diagonalization

We now ask: are there uncomputable functions? In terms of decision problems, are there non-r.e. languages? The answer is yes, but for a very fundamental reason that has almost nothing to do with computation (or so it seems). It has more to do with counting and size.

If X is a set, $|X|$ denotes its cardinality which is intuitively the number of its elements. This is clear for finite sets, as $|X| \in \mathbb{N}$. If X is infinite, we can simply say $|X| = \infty$. But we will want to refine this classification for infinite sets. The official way is via a theory of cardinals, so that the function $|\cdot|$ assigns a **cardinal number** $|X|$ to each set X. Without launching into a development of this theory, we can prove some basic facts via an axiomatic approach. Whatever we mean by "cardinal numbers", we want the following axioms to hold. Let X and Y be sets.

(A0) The cardinal numbers are totally ordered.

(A1) $|X| \leq |Y|$ iff there is an injection $f : X \to Y$.

From these two principles, we can conclude that if there is a bijection $h : X \to Y$ then $|X| = |Y|$. We also say X and Y are **equivalent** in this case. The converse is not so obvious, and is the subject of the celebrated Bernstein-Schröder theorem next. The quickest proof uses a fixed-point approach (this proof appears, for instance, in fixed-point semantics of of programming languages). The fixed-point principle is this:

Lemma 5 *Suppose $\mu : 2^X \to 2^X$ is* **monotone** *in the sense that $A \subseteq B \subseteq X$ implies $\mu(A) \subseteq \mu(B)$. Then there exists a set $A^* \subseteq X$ such that $\mu(A^*) = A^*$, called a* **fixed point** *of μ.*

Proof. Call a subset $A \subseteq X$ is "good" if $A \subseteq \mu(A)$. Define A^* to be the union of all the good sets. To show that A^* is a fixed point, we first show one direction:

$$A^* \subseteq \mu(A^*). \tag{2}$$

This amounts to saying that A^* is good. If $a \in A^*$ then $a \in A$ for some $A \subseteq A^*$ that satisfies $A \subseteq \mu(A)$. This shows that $a \in \mu(A)$. But $\mu(A) \subseteq \mu(A^*)$, by monotonicity. Thus $a \in \mu(A^*)$. In the other direction, we make a general observation: if A is good, then $\mu(A)$ is also good. This is because the goodness of A implies $\mu(A) \subseteq \mu(\mu(A))$, by monotonicity. Since A^* is good, $\mu(A^*)$ must be good. As A^* is the union of all good sets, this shows $\mu(A^*) \subseteq A^*$. **Q.E.D.**

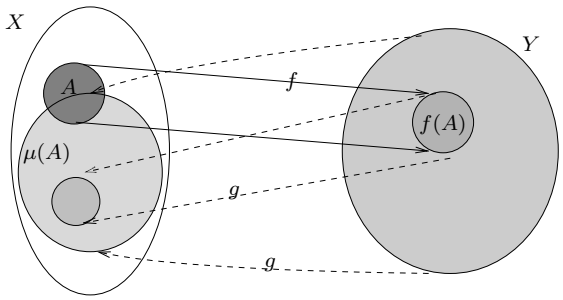

Figure 2: Injections $f : X \to Y$ and $g : Y \to X$.

Theorem 6 (Bernstein-Schöder) *If $f : X \to Y$ and $g : Y \to X$ are injections, then there exists a bijection $h : X \to Y$.*

Proof. First define the function $\mu : 2^X \to 2^X$ via

$$\mu(A) = X \setminus g(Y \setminus f(A)).$$

This function is monotone because if $A \subseteq B \subseteq X$, then

$$\begin{aligned} f(A) &\subseteq f(B) \\ Y \setminus f(A) &\supseteq Y \setminus f(B) \\ g(Y \setminus f(A)) &\supseteq g(Y \setminus f(B)) \\ X \setminus g(Y \setminus f(A)) &\subseteq X \setminus g(Y \setminus f(B)) \end{aligned}$$

The last inclusion amounts to $\mu(A) \subseteq \mu(B)$. By the previous lemma, μ has a fixed point A^*. Then $A^* = \mu(A^*) = X \setminus g(Y \setminus f(A^*))$. Writing $B^* := g(Y \setminus f(A^*))$, it follows that (A^*, B^*) is a partition of X. We now specify the bijection function $h : X \to Y$ as follows: h restricted to A^* is f, and h restricted to B^* is g^{-1}. To see that h is a bijection, it suffices note that $(h(A^*), h(B^*)) = (f(A^*), Y \setminus f(A^*))$ and hence forms a partition of Y as well. **Q.E.D.**

Let the cardinal number of $\mathbb{N}$ be denoted $\aleph_0$ ($\aleph$ is the first Hebrew letter **aleph**). A set X is said to be **countable** if there is an injection $f : X \to \mathbb{N}$. Equivalently, $|X| \in \mathbb{N}$ or $|X| = \aleph_0$. So Cantor's theorem proves that there are uncountable sets. The countability definition gives us one extra number (namely $\aleph_0$) to count with. We can at least distinguish among some infinite sizes. So let us try out out new tool.

It is easy to construct bijections between $\mathbb{N}$ and the sets $\mathbb{Z}$: For instance, if we list the elements of $\mathbb{Z}$ as $(0, -1, 1, -2, 2, -3, 3, \ldots)$, then we see that this listing can be matched to the listing $(0, 1, 2, 3, \ldots)$ of $\mathbb{N}$. Formally, the bijection is given by

$$f(n) = \begin{cases} n/2 & \text{if } n \text{ is even,} \\ -(n+1)/2 & \text{if } n \text{ is odd.} \end{cases} \tag{3}$$

Hence $|\mathbb{Z}| = \aleph_0$. This simple example illustrates a curious phenomenon that is unique to infinite sets: the property of having a bijection with a proper subset. Let us prove two more such results:

Lemma 7 *The following sets have cardinality* $\aleph_0$*:*
(a) The set Σ^* *of words over any alphabet* Σ*,*
(b) The set of all finite subsets of $\mathbb{N}$*.*

Proof. (a) We can list all the words of Σ^* in non-decreasing order of their lengths.
(b) Let $\mathbb{N}(k)$ denote all subsets of $\{0, 1, \ldots, k\}$. We simply list all the elements of $\mathbb{N}(0)$, then $\mathbb{N}(1) \setminus \mathbb{N}(0)$, $\mathbb{N}(2) \setminus \mathbb{N}(1)$, etc. **Q.E.D.**

Since we can view $\mathbb{Q} \subseteq 2^{\mathbb{N}}$ (each element of $\mathbb{Q}$ is just a pair of relatively prime numbers in $\mathbb{N}$), we conclude that $|\mathbb{Q}| = \aleph_0$ as well. At this point, the reader may have reason to worry that perhaps there are no other infinite cardinal numbers besides $\aleph_0$. The next theorem dispels this fear.

Theorem 8 (Cantor) *For any set* X*,* $|X| < \left|2^X\right|$.

Proof. Clearly $|X| \leq \left|2^X\right|$. Suppose, by way of contradiction, $f : X \to 2^X$ is a bijection. Let $D = \{x \in X : x \notin f(x)\}$. Hence there is $y \in X$ such that $f(y) = D$. We ask the question: is $y \in D$? If $y \in D$, then $y \in f(y)$ and so by definition of D, $y \notin D$. Conversely, if $y \notin D$, then $y \notin f(y)$ and so by definition of D, $y \in D$. Since both possibilities lead to contradiction, we have achieved a contradiction. **Q.E.D.**

This shows that in lemma 7(b), the restriction to finite subsets of $\mathbb{N}$ is essential. It is a fact that $\left|2^X\right| = 2^{|X|}$. Hence, $2^{\mathbb{N}}$ has a cardinality $2^{\aleph_0}$ which this theorem shows is different from $\aleph_0$. The famous **continuum hypothesis** says that there does not exist a cardinal number α such that $\aleph_0 < \alpha < 2^{\aleph_0}$. Put another way, if $\aleph_1$ denotes the smallest cardinal number larger than $\aleph_0$, the hypothesis asserts that $\aleph_1 = 2^{\aleph_0}$.

Cantor's theorem uses a **diagonalization argument**. Why is this so-called? Imagine a semi-infinite matrix indexed by X on the rows and 2^X on the column. D is constructed by looking at the diagonal of the matrix. Diagonalization turns out to be a very important technique in the theory of computability.

Cantor's theorem shows that there exists uncountable sets. The set $\mathbb{R}$ of real numbers is a familiar set that is uncountable: We only need to prove that the real numbers in the interval $[0, 1]$ is uncountable. We use the fact that each such numbers has an infinite binary expansion of the form $0.b_1b_2b_3\cdots$. If $[0, 1]$ is countable, we can arrange them in an infinite list $(x_1, x_2, x_3, \ldots)$. Consider a new number r whose decimal expansion is given by $0.d_1d_2d_3\cdots$ where d_n is chosen to be different from the nth bit of x_n. Note that this number is different from any in the list, and this is a contradiction. This sketch is not yet complete: the binary notation is non-unique. The binary strings $0.b_1\cdots b_n10^\omega$ and $0.b_1\cdots b_n01^\omega$ represents the same number. One solution is to use a m-ary notation that uses the digits $\{0, 1, \ldots, m-1\}$ for any $m \geq 4$. For instance, with quaternary numbers ($m = 4$), the numbers $0.t_1t_2\cdots t_{n-1}t_n3^\omega$ and $0.t_1t_2\cdots t_{n-1}(t_n + 1)0^\omega$ are equal (here $t_n \in \{0, 1, 2\}$). But in our construction of the number r, we only choose the digits 1 or 2 to ensure that r has a unique representation.

Theorem 9 *There is a non-r.e. language.*

Proof. This theorem follows from two simple claims. and $\mathcal{L}$ the set of languages.
CLAIM A: The class of all languages $\mathcal{L}$ is uncountable.
CLAIM B: The family of all Turing machines $\mathcal{M}$ is countable.
Assuming these claims, and if every language were r.e., we obtain a contradiction. For, there is an injection $f : \mathcal{M} \to \mathcal{L}$ where $f(M)$ is just $L(M)$. If the theorem were false, then f is a surjection. Then let $g : \mathcal{L} \to \mathcal{M}$

be the injection defined such that $g(L) \in f^{-1}(L)$ (we need the axiom of choice here). But this g contradicts contradicts the claims that asserts $|\mathcal{L}| > |\mathcal{M}|$.

Proof of CLAIM A: let $\mathcal{L}_0$ be the set of languages over 0/1. It suffices to show this set uncountable. But clearly $\mathcal{L}_0$ is equivalent to $2^{\mathbb{N}}$. By Cantor's theorem, this set is uncountable.

Proof of CLAIM B: CLAIM A is not literally true. If the set of allowed symbols is uncountable, $\mathcal{M}$ is trivially uncountable. We can do is this: restrict the states and tape alphabet of our Turing machines to be finite subsets of $\mathbb{N}$. Call this restricted class of machines $\mathcal{M}_0$ (the "nice" TM's). It is not hard to show that $\mathcal{M}_0$ is now countable: let $\mathcal{M}_0(n)$ comprise all the STM's whose transition table uses the state or symbol n, but no states or symbols larger than n. Then we first list all the machines in $\mathcal{M}_0(0)$, then $\mathcal{M}_0(1)$, etc. The function $f : \mathcal{L}_0 \to \mathcal{M}_0$ can be defined as before, yielding a contradiction. **Q.E.D.**

Universal Assumptions. The proof of the preceding theorem motivates the following convention. Henceforth, we make two universal[3] assumptions: Let Σ_∞ and Q_∞ be two disjoint infinite sets.

(α) Σ_∞ contains all alphabets of our languages. The blank symbol $\sqcup \in \Sigma_\infty$ is special, and belongs to no alphabet.

(β) Q_∞ contains all the states of our Turing machines. The states $q_0, q_A \in Q_\infty$ are special, representing the start and accepting states of all Turing machines.

The main advantage of these assumptions is that we can now assert that *the family $\mathcal{M}$ of all STM's is countable*. The practical advantage is that we save verbiage in discussing Turing machines. Instead of a 6-tuple, a STM M is now simply a finite set δ_M of the form

$$\delta_M \subseteq Q_\infty \times \Sigma_\infty \times \{-1, 0, +1\} \times Q_\infty \times \Sigma_\infty.$$

That is, we need not explicitly specify the state set Q and alphabet Σ as these can be directly deduced from δ_M. Similarly, the start q_0 and accept q_A states need not be specified as they are universally chosen.

§4. Universal Machines

We introduce the **universal Turing machine** (UTM). A UTM is similar to a Simple Turing machine U but it has two tapes. One tape has the same properties as the tape of the STM, and is called the **work tape**. The other tape is called the **index tape**. There are corresponding **work head** and **index head** that is scanning a cell on the respective tapes. The index tape is a **read only** tape (writing not allowed) and its alphabet is restricted to $\{0, 1, \sqcup\}$. The input to the UTM is a pair $\langle i, x\rangle$ of words where $i \in \{0,1\}^*$. Intuitively, i is the number of a STM. The transition table δ_U of U is a finite set

$$\delta_U \subseteq Q_\infty \times \Sigma_\infty \times \{0, 1, \sqcup\} \times Q_\infty \times \Sigma_\infty \times \{-1, 0, 1\}^2.$$

A typical instruction of δ_U is

$$(q, a, b \to q', a', D_1, D_2)$$

with precondition that the machine is in state q, scanning a on the work tape, and scanning b on the index tape. Upon executing the instruction, the new state is q', a is changed to a' and the work head and index head moves in directions D_1, D_2 (respectively). The machine begins in the initial state q_0 and accepts its input if there exists a computation path that leads to the accept state q_A. As usual, $L(U) = \{\langle i, x\rangle : U \text{ accepts } \langle i, x\rangle\}$.

[3]The assumptions are also "universal" in a technical sense related to the concept of universal Turing machines.

Note that $L(U)$ is no longer a language, but a binary relation. We may interpret the index i as a dyadic number and thus regard $L(U)$ as a subset of $\mathbb{N} \times \Sigma^*$ for some Σ^*. Define the language class $K(U) := \{L_i(U) : i \in \mathbb{N}\}$ where $L_i(U) := \{w \in \Sigma^* : \langle i, w\rangle \in L(U)\}$, We say U **accepts the class** $K(U)$ of languages. It is important to note that $K(U)$ is a collection of languages over some fixed alphabet $\Sigma \subseteq \Sigma_\infty$. Write $K|\Sigma$ to denote the restriction of a class K to those languages over Σ. Thus $K(U) = K(U)|\Sigma$ for some Σ. Typically, $\Sigma = \{0, 1\}$ in this notation.

Most of the definitions for STM is transferred naturally to the new setting. Thus it is clear what we mean by a deterministic UTM. We write write $U(i, w) \uparrow$ or $U(i, w) \downarrow$ depending on whether U loops or halts on $\langle i, w\rangle$.

Theorem 10 (Universal TM) *Fix any alphabet $\Sigma \subseteq \Sigma_\infty$. There is a universal Turing machine U such $K(U) = RE|\Sigma$. Furthermore:*

- *U is deterministic and does not write on its index tape.*
- *Let $\mathcal{M}$ be the family of STM's over input alphabet Σ. There is an index function $\iota : \mathcal{M} \to \mathbb{N}$ such that for all $M \in \mathcal{M}$ and $w \in \Sigma^*$, $U(\iota(M), w) = M(w)$. and $L_{\iota(M)}(U) = L(M)$.*

Proof. The function $\iota : \mathcal{M} \to \mathbb{N}$ encodes each STM M as a dyadic number[4] $\iota(M) = i$. In case i is not the encoding of an STM, we can simply ignore what U actually accepts (but we know it is some r.e. language). Alternatively, if our encoding allows it, we can detect when i is not a legitimate encoding and immediately reject the input. We can choose an encoding that allows one to read off the set of instructions of M_i in a straightforward manner (Exercise).

On input $\langle i, w\rangle$ machine U simulates the machine $f^{-1}(i)$ on w, deterministically. To do this, it stores a list of configuration of $f^{-1}(i)$ on its work tape. This follows the proof of theorem 1. The U behaves as in that proof, except that to carry out an expansion of a configuration, it must use the index tape to look up the next executable instructions. Since U is simulation M_i, **Q.E.D.**

From now on, let $U^{(2)}$ denote the universal Turing machine guaranteed by this theorem, with $\Sigma = \{0, 1\}$. For $M \in \mathcal{M}$, if $\iota(M) = i$, we shall write

$$U_i^{(2)} \sim M.$$

§5. Decision Problems

If L is any language, the computational problem of designing an algorithm to accept L is called the **decision problem** for L. We introduce several decision problems related to STMs. Among them we will find non-r.e. languages that are considered "natural", unlike the artificial (non-explicit) ones obtained by cardinality arguments. All these problems will exploit the existence of universal machines.

Three Decision Problems. Let $U^{(2)}$ be the UTM in the theorem 10, with $\Sigma = \{0, 1\}$. Define the languages

$$W_2 \quad = \quad \{i\#w : i, w \in \{0, 1\}^*, \langle v, w\rangle \in L(U^{(2)})\}. \tag{4}$$

[4]Binary numbers are non unique: e.g., 011 and 11 both represent the number 3. A dyadic number is rather similar to binary numbers, but they are unique. If $w = b_n b_{n-1} \cdots b_1 b_0$ ($b_i \in \{0, 1\}$), then, as a dyadic number w, represents the integer $\#(w) = \sum_{i=0}^{n} (b_i + 1)2^i$. Thus, $\#(\epsilon) = 0$ and $\#(01) = 4$. One checks that this function $\# : \{0, 1\}^* \to \mathbb{N}$ is a bijection. Sometimes, instead of the alphabet $\{0, 1\}$, one uses $\{1, 2\}$ for dyadic notation. We prefer to stick to $\{0, 1\}$ to avoid the unfamiliar appearance of strings such as 12112.

$$D_2 = \{w \in \{0,1\}^* : \langle w, w\rangle \in L(U^{(2)})\}. \quad (5)$$
$$H_2 = \{w \in \{0,1\}^* : U^{(2)}(\langle w, w\rangle) \downarrow\}. \quad (6)$$
$$(7)$$

The decision problems for W_2, D_2, H_2 are called (respectively) the **word problem**, the **diagonal problem** and the **halting problem** for $U^{(2)}$.

Theorem 11 *$W_2, D_2, H_2 \in RE$.*

Proof. First we show $W_2 \in RE$. This amounts to constructing a STM M_2 that emulates $U^{(2)}$. On input $i\#w$, M_2 simulates the actions of $U^{(2)}$ on input $\langle i, w\rangle$ in a step-by-step manner. M will accept iff $U^{(2)}$ accepts. Thus $L(M_2) = W_2$. Similarly, D_2 and H_2 can be shown to be in RE be simple variations of this simulation. **Q.E.D.**

Theorem 12 *$W_2, D_2, H_2 \notin REC$.*

Proof. Suppose W_2 is decided by a deterministic halting STM M. We construct another halting STM M' such that M' does the opposite of M:

M' on input i:
1. M' runs M on $i\#i$. 2. M' accepts iff M' rejects.

Note that M' is halting since M is halting. Since M' can clearly be constructed from M, we conclude that $M' \sim U_j^{(2)}$ for some j. Now consider what happens when M' is given the input j. (a) If M' accepts j, it must be because M rejects $j\#j$. By definition of M, this means $U^{(2)}(j, j)$ rejects. But this means M' rejects j, contradiction. (b) If M' rejects j, it must be because M accepts $j\#j$. By definition of M, this means $U^{(2)}(j, j)$ accepts. But this means M' accepts j, contradiction.

Q.E.D.

Theorem 13 *co-W_2, co-D_2, co-H_2 are not r.e..*

Proof. We use the fact that $REC = RE \cap \text{co-}RE$. If co-$W_2 \in RE$, then W_2 would be recursive, contradicting the previous theorem. The other two cases are similar. **Q.E.D.**

Rice's Theorem. There is general method to show undecidable problems. Let K be any class of languages. Relative to $U^{(2)}$, we define the language

$$L(K) := \{i \in \{0,1\}^* : L_i(U) \in K\}.$$

Theorem 14 (Rice) *If $K \neq \emptyset$ is a proper subset of RE, the $L(K)$ is not recursive.*

Proof.

Q.E.D.

§6. NOTES on Cardinal and Ordinal numbers

How do we compare the "sizes" of sets X and Y? When X and Y are finite sets, we can list all the elements of X and Y in some "listing order" $(x_0, x_1, \ldots)$ and $(y_0, y_1, \ldots)$ and to match them up $x_0 \leftrightarrow y_0$, $x_1 \leftrightarrow y_1$, etc. The list that runs out of elements first has smaller size; if both list runs out of elements simultaneously, we say that they have the same "size". This "matching principle" is clearly valid in the finite case, as the outcome is independent of the listing order we choose. For infinite sets, this matching principle calls for refinement. For instance, the listing orders that show $|\mathbb{N}| = |\mathbb{Z}|$ (see (3)) must be carefully chosen or it may not work.

The concept of listing order generalizes to the concept of **ordinal numbers**. To get a matching principle that is independent of listing order, we need the concept of **cardinal numbers**. Informally, the relationship between these two concepts is seen in English where the "cardinal numbers" `zero`, `one`, `two`, etc., are used to count, while the "ordinal numbers" `zeroth`, `first`, `second`, etc., are used to order or rank. The theory of cardinals will be built on ordinal theory.

Define an **ordinal** to be a well-ordered set α such that for every $x \in \alpha$, the set $s(x) = \{y \in \alpha : y < x\}$ also belongs to α. The set $s(x)$ is called an **initial segment** of α. Any two ordinal numbers α, β can be compared. This follows from general considerations about well-ordered sets. Two partially ordered sets $(A, \leq)$, $(A', \leq')$ are **similar** if exists a bijection $h : A \to A'$ that is order preserving (i.e., $a \leq b$ implies $h(a) \leq' h(b)$. If A, A' are well ordered, then we write $A \prec A'$ if A is similar to an initial segment of A'. This is clearly a transitive relation. For well ordered sets A and A', exactly one of the following holds:

$$A \sim A', \qquad A \prec A', \qquad A' \prec A.$$

See for example [1, p. 73]. This is the total ordering we use for ordinal numbers.

To view the natural numbers $n \in \mathbb{N}$ as ordinals, we must interpret them as sets using the following device: if $n = 0$, then n is the empty set; otherwise n is the set $\{0, 1, \ldots, n-1\}$. Thus

$$0 = \emptyset, \quad 1 = \{\emptyset\}, \quad 2 = \{\emptyset, \{\emptyset\}\}, \quad 3 = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}, \quad \ldots.$$

Now we may verify that the set $\mathbb{N}$ is itself an ordinal number! This ordinal number is denoted ω. This is the first infinite ordinal. Given any ordinal α, we can define its **successor** α^+, which is the ordinal $\alpha \cup \{\alpha\}$, For instance, $\omega^+ = \mathbb{N} \cup \{\mathbb{N}\}$. Moreover, the operations of addition, multiplication and exponentiation can be defined. But be prepared for surprises: the ordinals $1 + \omega$ and $\omega + 1$ may be represented as

$$1 + \omega = \{0', 0, 1, 2, 3, \ldots\} \qquad \omega + 1 = \{0, 1, 2, 3, \ldots, \omega\},$$

and this yields the unintuitive result, $1 + \omega = \omega < \omega + 1$. Similarly, $2\omega = \{0', 0, 1', 1, 2', 2, 3', \ldots\}$ and $\omega 2 = \{0', 1', 2', \ldots, 0, 1, 2', \ldots\}$, and thus $2\omega = \omega < \omega 2$.

Since distinct ordinal numbers can have the same cardinality (i.e., equivalent), we define a **cardinal number** to be smallest ordinal number among its equivalence class. Thus ω is a cardinal number and is in fact what we denoted by $\aleph_0$ before. But $\omega + 1$ and 2ω are not cardinal numbers as they are equivalent to ω. Cardinal comparison is induced from ordinal comparison. Again, cardinal arithmetic can be defined.

BACKGROUND NOTES

The proofs that W_2, D_2, H_2 are not recursive is basically a diagonal argument. This is also the method by which one proves Gödel's famous incompleteness theorem: that there exists a true statement of number theory that cannot be proven. What is needed, and this is Gödel's remarkable achievement, is a way to encode provability and statements of number theory within number theory itself. Provability is akin to computability. Diagonal arguments appear at the foundation of set theory as paradoxes (contradictions). These paradoxes typically has some self-referential element. For instance: `In a certain village, there are many barbers. But there is one barber who shaves any barber who does not shave himself. Who shaves this barber?` Pondering the two possibilities, one is led to the conclusion that this barber does not exist. In the set theoretic paradoxes, we can similarly define sets that do not exist. This led to efforts to axiomatize set theory, basically to carefully prescribed circumstances under which new sets can be legitimately formed. Happily, all the operations we make in these notes are admissible (power set is admissible for instance).

Exercises

Exercise 6.1: For any alphabet $\Sigma \neq \emptyset$, construct a bijection between Σ^* and $\mathbb{N}$ and prove that it is indeed a bijection. ♢

Exercise 6.2: Prove that if A is r.e. but not recursive, then it contains an infinite subset that is recursive. ♢

Exercise 6.3: Fix the proof that the reals in $[0, 1]$ is uncountable, but remain in binary notation. Alternatively, you can use dyadic notation. ♢

Exercise 6.4: (i) Show that L is r.e. iff L is generated by some deterministic STM.
(ii) Show that L is recursive iff L is generated by some deterministic STM in lexicographic order. ♢

End Exercises

END OF LECTURE

References

[1] P. R. Halmos. *Naive Set Theory.* Van Nostrand Reinhold Company, New York, 1960.

[2] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 2(42 and 43):230–265 and 544–546, 1936.