Notes on Computer Theory Last updated: September, 2015

Diagonalization, halting problem, reducibility

Notes taken from Jonathan Katz, lightly edited by Dov Gordon

1 Diagonalization

Last time, we proved that the following language is undecidable:

$$L_{halt} = \{(\langle M \rangle, x) \mid M(x) \text{ halts with output } 1\}$$

We proved that through a contradiction: assuming that L_{halt} is decided by turing machine M_{halt}, we showed that the following turing machine M^*, which is well formed, has undefined output:

> M^*: On input (a description of) a Turing machine $\langle M \rangle$, compute $M_{halt}(\langle M \rangle, \langle M \rangle)$. If the result is 1, output 0; otherwise output 1.

Today we will give an alternative perspective on the same proof by describing this as a an example of a *general* proof technique called diagonalization. This techniques was introduced in 1873 by Georg Cantor as a way of showing that the (infinite) set of real numbers is larger than the (infinite) set of integers. We will define what this means more precisely in a moment.

Definition 1 *Given two sets, A and B, and a function $f : A \to B$, we say that f is* injective *(one-to-one) if it never maps two different elements from A onto the same element from B: $\forall a, b \in A, a \neq b \Rightarrow f(a) \neq f(b)$*

Definition 2 *Given two sets, A and B, and a function $f : A \to B$, we say that f is* surjective *(onto) if every value in B has a pre-image under f: $\forall b \in B, \exists a \in A$ s.t. $f(a) = b$.*

If a function is both one-to-one and onto, then we say it is *bijective*, or a *correspondence*. Cantor proposed that we can use a correspondence to compare the sizes of infinite sets, the same way would finite sets. In particular, we are interested in determining whether some infinite set is the same size, or larger than the set of natural numbers $\mathcal{N} = \{1, 2, 3, \ldots\}$. If a set S has a correspondence with the natural numbers, i.e. $f : \mathcal{N} \to S$, we say that the set is *infinitely countable*.

Consider, for example, the set of even numbers $\{2, 4, 6, \ldots\}$. We might think that this set is smaller than the set of natural numbers, since it is fully contained within the set of natural numbers. But we can also see that there is a correspondence mapping from $\mathcal{N}$ to the set of even numbers: $f(a) = 2a$. (Verify for yourself that f is both one-to-one and onto.)

How about the set of positive rational numbers $\mathcal{Q} = \{\frac{m}{n} \mid m, n, \in \mathcal{N}\}$? Our first intuition might be that this set is "larger" than $\mathcal{N}$, but in fact we can again build a correspondence between the two sets. To do that, let's start by building a matrix of all positive rational numbers, placing any number with i in the numerator in the ith row, and any number with j in the denominator in the

jth column.

$$\begin{pmatrix} \frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots \\ \frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \frac{2}{4} & \frac{2}{5} & \cdots \\ \frac{3}{1} & \frac{3}{2} & \frac{3}{3} & \frac{3}{4} & \frac{3}{5} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Then, we go through the array, mapping each new item to one of the natural numbers. However, when doing that, we don't simply iterate over them row by row, or column by column, because each row and column has infinite size, and we'd never get past the first row or column! Instead, we move through the matrix along diagonals that go up from left to right: $\frac{1}{1}, \frac{2}{1}, \frac{1}{2}, \frac{3}{1}, \frac{1}{3}, \frac{4}{1}, \frac{3}{2}, \frac{2}{3}, \frac{1}{4}, \frac{5}{1}, \frac{1}{5}$. Note that in order to ensure that our mapping is one-to-one, we have to skip any repeated values. So, for example, we skip $\frac{2}{2}$, because we already included $\frac{1}{1} = \frac{2}{2}$. We also skipped $\frac{4}{2}, \frac{3}{3}$ and $\frac{2}{4}$ for the same reasons. The correspondence, f, simply maps the $a \in \mathcal{N}$ onto the ath element on this list. f is one-to-one because we have only included one "copy" of each equivalent rational number on the list, so no two natural numbers are mapped onto the same rational number. f is onto because there is no (positive) rational number left off of our list.

Theorem 1 *$\mathcal{R}$ is not countable.*

Proof We show this by contradiction. Suppose that $\mathcal{R}$ were countable, and let f be the correspondence with $\mathcal{N}$. We will show that there exists some $b \in \mathbb{R}$ that is not in the image of f, violating the assumption that f is onto. To do this, we start by ordering all the elements of $\mathbb{R}$ according to the values of their pre-images: $f(1), f(2), f(3), f(4), \ldots$. Because f is a correspondence, *all* elements of $\mathbb{R}$ must appear somewhere on this list. We now define a real number $b \in \mathbb{R}$ that does not appear on this list, demonstrating a contradiction. Recall that real numbers can be infinitely long. Choose the ith digit of b such that it is different from the ith digit of $f(i)$. Suppose that b appears in the jth position of this list (i.e. that $f(j) = b$). But this cannot be true, because we know that b differs from $f(j)$ in the jth position! This is called diagonalization, because we define the contradictory value by appropriately setting its value along the diagonal. ■

We now return to the halting problem. To prove that L_{halt} is not decidable, we begin by assuming that it M_{halt} decides it, and we list the output of this machine on all possible inputs. Recall that it takes two inputs: $\langle M \rangle$ and w, where the first is the description of a turing machine, and the second is interpreted as the input to M. We will list the outputs of M_{halt} by listing all possible Turing machines on one side of a matrix, and all of their descriptions on the other side of a matrix. (Note that there is a correspondence mapping the set of Turing machines to the set of finite-length strings, so the second input to M_{halt} can always be interpreted as the description of some Turing machine. However, this point is not essential for the proof: we're simply only interested in the strings that represent Turing machines and therefore only list those along the top row.)

We can now list the output of M_{halt}:

$$\begin{pmatrix} & \langle M_1 \rangle & \langle M_2 \rangle & \langle M_3 \rangle & \langle M_4 \rangle & \dots \\ M_1: & accept & accept & reject & reject & \dots \\ M_2: & accept & reject & reject & reject & \dots \\ M_3: & reject & accept & reject & accept & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

We defined M^* to run $M_{halt}(M, M)$ and do the opposite. Since M^* is a valid turing machine, it must appear somewhere on the list we described above. We fill in its output in the table below:

$$\begin{pmatrix} & \langle M_1 \rangle & \langle M_2 \rangle & \langle M_3 \rangle & \langle M_4 \rangle & \dots & \dots \\ M_1: & \underline{accept} & accept & reject & reject & \dots & accept \\ M_2: & accept & \underline{reject} & reject & reject & \dots & reject \\ M_3: & reject & accept & \underline{reject} & accept & \dots & accept \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & reject \\ M^*: & reject & accept & accept & reject & \dots & \underline{???} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

1.1 Unrecognizable languages

The contradiction we just showed demonstrates that the language L_{halt} is not decidable. We can actually prove that there exist languages that aren't even *recognizable*. Recall, the difference is that a language is recognizable if a turing machine accepts every string in the language, but has undefined behavior on strings that are not in the language (and, in particular, might run forever on such strings).

We show this by proving that there are more languages than Turing machines: the set of Turing machines is countable, but the set of languages is not! We use the following lemmas and corollaries in our proof.

Lemma 2 *Any set $X \subseteq \mathcal{N}$ is countable.*

Corollary 3 *For any alphabet Σ, the set of strings Σ^* is countable.*

Corollary 4 *The set of all valid Turing machines is countable.*

Lemma 5 *The set of infinite binary sequences is uncountable.*

The proof of this lemma is essentially identical to the proof that $\mathbb{R}$ is not countable, so we omit it. Note that the key difference between the statement of Corollary **??** and this one is that is that no string in the set described in Corollary **??** can have infinite length, even though we place no limit on the size that any string in that set *can* have. In other words, for any string $w \in \Sigma^*$, there exists some constant c such that $w = \Sigma^c$.

Theorem 6 *There exist languages that are not Turing recognizable.*

Proof We already observed that the set of all Turing machines is countable. To show that the set of all languages is not countable, we give a correspondence to the set of infinite binary strings, and rely on the second lemma above to complete the proof. To describe this correspondence, we define a function f mapping each infinite binary string onto some language. We start with a fixed ordering on all binary strings: we list all strings of length 1 in lexicographic order, followed by all strings of length 2 in lexicographic order, etc. Then, for some infinite binary string w, we define the language $f(w)$ by including the jth binary string on this list in the language $f(w)$ if and only if the jth bit of w is 1. To see that f is one-to-one, consider two infinite binary strings $w \neq w'$, and, suppose they differ on the jth bit. $f(w) \neq f(w')$, since the two languages differ on whether they include the jth string from our canonical list of all strings. It is easy to verify that every language has a pre-image under f by simply defining the inverse function: given some language L, we can find w such that $f(w) = L$ by starting with the same canonical ordering on all strings, and letting the jth bit of w be 1 iff the jth string is in L.

Since there are more languages than there are Turing machines, it follows, at least intuitively, that there exists some language that is not recognized by any Turing machine. To formalize this, we actually require an additional lemma.

Lemma 7 *Let S be some infinite set, and let f be a surjective (onto) function $f : \mathcal{N} \to S$. Then there exists a bijection $g : \mathcal{N} \to S$.*

Proof f is onto S, but it is not necessarily one-to-one, which means there exist $a, b \in \mathcal{N}$ s.t. $a \neq b$ and $f(a) = f(b)$. Intuitively, we want to "re-map" f on one of these values so that they no longer collide. Since we have infinite space to deal with, we can simply choose the next available value, and "shift" everything down. More formally, we define g inductively: let $g(1) = f(1)$, and for each $i > 1$, assume that $g(i-1)$ has already been defined. Let $j \in N$ be such that $f(j) = g(i-1)$. To define $g(i)$, take the smallest value $j' > j$ such that $\forall x < j', f(x) \neq f(j')$. Set $g(i) = f(j')$.

We need to show that g is a bijection. To see that it is injective, assume that there are no collisions up until the inductive step: i.e. $\forall x, y < i, g(x) \neq g(y)$. Then the assumption still holds after the inductive step, since $g(i) = f(j')$, where j' was specifically chosen such that $\forall x < j', f(x) \neq f(j')$.

To see that it is still surjective, consider any $x \in S$, and let i be some value such that $f(i) = x$. (Because f is surjective, we know that some such value exists.) If for some $j < i, g(j) = x$, then we're done. If $\forall j < i, g(j) \neq x$, then, when choosing $g(i)$, it holds that $f(1), \ldots, f(i-1)$ are already in the image of g, and i is the smallest value such that $\forall j < i, f(j) \neq f(i)$. By the definition of g, it follows that $g(i) = f(i) = x$. ■

Consider the obvious function mapping the set of Turing machines onto the set of recognizable languages. (This function is not a correspondence — can you see why? — but it will suffice for our proof.) This function cannot be onto the set of *all* languages, or else, by the previous lemma,

it would follow that the set of languages is countable. It follows that there exists some language without any pre-image under this function; this language is not recognized by any Turing machine. ■

We give one more proof of the same theorem. Actually, in this proof, we will identify a particular language that is not recognizable. (In the previous proof, we only proved that some such language exists, but we were not able to identify an example.) Consider the language $\overline{L}_{halt}$, which is the complement of L_{halt}. In other words, $\overline{L}_{halt} = \{\langle M \rangle, x \mid M \text{ does not halt and output 1 on input } x\}$. We will prove that this language is not recognizable by any Turing machine. We begin with the following theorem.

Theorem 8 *A language is decidable if and only if both it and its complement are Turing recognizable.*

Proof Let L be some decidable language. It follows immediately from the definitions that L is recognizable. To see that $\overline{L}$ is also recognizable, let M_L be the machine that decides L, and define $\overline{M}_L$ as the machine that runs M and reverses its output. In the other direction, suppose L and $\overline{L}$ are recognizable by M_L and $\overline{M}_L$ respectively. To decide L, we can construct a Turing machine that runs both of these machines in parallel, alternating the steps of each computation. (The alternation of steps is important, in case either one of these machines never terminates.) On input x, if $M_L(x)$ halts with output 1, then halt and output 1. If $\overline{M}_L$ halts and outputs 1, halt and output 0. ■

We have proven that L_{halt} is not decidable. However, note that it is recognizable, since on input $(\langle M \rangle, x)$ we can always simulate M on x: if M halts and outputs 1, then we do the same. (The fact that it is not decidable means that it might not halt when M does *not* accept x; in this case, our simulation would not halt either.) It follows that $\overline{L}_{halt}$ is not recognizable: otherwise, by the previous theorem, L_{halt} would be decidable.

2 Reducibility

Consider the following language $L_{A/R} = \{\langle M \rangle, x \mid M \text{ halts on input } x \text{ and outputs either 0 or 1}\}$. This looks very similar to L_{halt}, which we know is undecidable, so it is not surprising that this language is also undecidable. But how do we prove that? We can do it through a *reduction*: we demonstrate that if there is a Turing machine $M_{A/R}$ that decides $L_{A/R}$, then there is a Turing machine M_{halt} that decides L_{halt}. Since we already know that the latter statement is untrue, it follows that the former statement is untrue.

Theorem 9 *The language $L_{A/R}$ is undecidable.*

Proof We prove it by constructing M_{halt} that does the following.
$M_{halt}(\langle M \rangle, x)$:

1. Simulate $M_{A/R}(\langle M \rangle, x)$.
   (a) If it outputs 0, halt and output 0.
   (b) If it outputs 1, simulate M on input x until it halts. Output whatever M outputs.

■

Theorem 10 *The language $L_\emptyset = \{\langle M \rangle \mid M \text{ is a Turing machine and } L(M) = \emptyset\}$ is undecidable.*

Proof We again assume that there exists some TM $M_\emptyset$ deciding $L_\emptyset$, and demonstrate that this gives rise to a TM M_{halt} that decides L_{halt}. We describe M_{halt} as follows.
$\underline{M_{halt}(\langle M \rangle, x):}$

1. Write down a description of a TM M' that modifies the behavior of M as follows.
   $\underline{M':}$
   (a) On input $y \neq x$, reject.
   (b) On input $y = x$, run $M(y)$ and output whatever it outputs.
2. Run $M_\emptyset(\langle M' \rangle)$. Reverse the value of its output.

■

Recall, last time we had the following definition:

Definition 3 *A language L is* Karp reducible *(or* many-to-one reducible*) to a language L' if there exists a polynomial-time computable function f such that $x \in L$ iff $f(x) \in L'$. We express this by writing $L \leq_p L'$.*

We can also drop the requirement that f be computable in polynomial time; such a reduction is sometimes called a *mapping reduction*. It can be expressed by $L \leq_m L'$ instead of $L \leq_p L'$.

We can use the reduction given above to construct a mapping reduction from L_{halt} to $L_{A/R}$, i.e. we can show that $L_{halt} \leq_m L_{A/R}$, again implying that if $L_{A/R}$ is computable, then so is L_{halt}. We need to give a computable function f such that

$$(\langle M \rangle, x) \in L_{halt} \Leftrightarrow f(\langle M \rangle, x) \in L_{A/R}$$

Note that f in this case outputs the description of a Turing machine and its input. Specifically, on input $(\langle M \rangle, x)$, f outputs a $(\langle M' \rangle, x)$, where M' does the following.
$\underline{M'(\langle M \rangle, x):}$

1. Simulate M on input x.
   (a) If it outputs 1, halt and output 1.
   (b) If it outputs 0, loop forever.

Let's try to do the same with the reduction we built from L_{halt} to $L_\emptyset$. $f(\langle M \rangle, x)$ should output the description of a Turing machine, M', such that, if $M(x)$ halts with output 1, then $M' \in L_\emptyset$ and rejects all strings. The natural way to try to define f follows along the lines of what we did above in our reduction: f should output M' that rejects on all strings $y \neq x$, and then runs $M(y)$ when $y = x$. Unfortunately, note that this actually gives a reduction to $\overline{L}_\emptyset$! And we cannot simply say that M' should reverse the output of $M(y = x)$, because it could be that M does not halt at all on this input, in which case M' runs forever and is not a decider for any language. In the homework, you will be asked to prove that no such mapping reduction exists. However, note that this does not mean our previous proof was incorrect! We *do* have a good reduction to $\overline{L}_\emptyset$, and since decidability is closed under complement, this suffices to prove that $L_\emptyset$ (as well as $\overline{L}_\emptyset$) is undecidable as well.

References

[1] M. Sipser. *Introduction to the Theory of Computation* (2nd edition). Course Technology, 2005.