Notes For CMSC 650- COMPUTABILITY
by A.Amir and W.I. Gasarch

1 Introduction

We take our model of computation to be Java Programs. Traditionally one defines Turing Machines and we will use that terminology, but they are really java programs. One thing to note: When we say something like

$M_1, M_2, \ldots,$ *is a standard list of Turing Machines*

we mean that this includes ALL programs and that, from the index i, you can extract code for i. In particular there is a Turing Machine M that takes as input (i, x) and outputs $M_i(x)$.

2 Computable and Computably Enumerable Sets

Def 2.1 A set A is *computable* if there exists a Turing Machine M that behaves as follows:

$$M(x) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{if } x \notin A. \end{cases}$$

Computable sets are also called decidable or solvable. A machine such as M above is said to decide A.

Some examples of computable sets.

1. The primes.
2. The Fibaonoacci numbers (any number in the set $1, 2, 3, 5, 8, 13, ...$ where every number is the sum of the previous number). If you want to know if a number x is a Fib number, just calculate the Fib numbers until you either spot x or surpass it. If you spot it then its a Fib number, if you surpass it, its not.
3. (x, y, s) such that $M_x(y)$ halts within s steps.
4. Most sets you can think of are computable.

Are there any noncomputable sets? Cheap answer: The number of SETS is uncountable, the number of COMPUTABLE SETS is countable, hence there must be some noncomputable sets. In fact, there are an uncountable number of them. I find this answer rather unenlightening.

3 The HALTING Problem

In this section we exhibit a concrete example of a set that is r.e. but not computable. Recall that M_x is the xth Turing Machine in the Godelization defined earlier.

Def 3.1 The HALTING set is the set

$$K_0 = \{\langle x, y\rangle \mid M_x(y) \text{ halts } \}.$$

Let us ponder how we would TRY to determine if a number $\langle x, y\rangle$ is in the halting set. Well, we could try RUNNING M_x on y. If the computation halts, then GOOD, we know that $\langle x, y\rangle \in K_0$. And if it doesn't halt then – WHOOPS– if it never halts we won't know that!! It seems hard to determine with certainty that the machine will NOT halt EVER.

Theorem 3.2 *The set K_0 is not computable.*

Proof:

We show that K_0 is NOT computable, by using diagonalization. Assume that K_0 is computable. Let M be the Turing Machine that decides K_0. Using M we can easily create a machine M' that operates as follows:

$$M'(x) = \begin{cases} 0 & \text{if } M_x(x) \text{ does not halt,} \\ \uparrow & \text{if } M_x(x) \text{ does halt.} \end{cases}$$

Since M' is a Turing Machine, it has a Godel number, say e, so $M_e = M'$. We derive a contradiction by seeing what M_e does on e.

If $M'(e) \downarrow$ then by the definition of M', we know that $M_e(e)$ does not halt, but since $M' = M_e$, we know that $M_e(e)$ does halt. Hence the scenario that $M'(e) \downarrow$ cannot happen. (This is not a contradiction yet)

If $M'(e) \uparrow$ then by the definition of M', we know that $M_e(e)$ does halt'; but since $M' = M_e$, we know that $M_e(e)$ does not halt. Hence the scenario that $M'(e) \uparrow$ cannot happen. (This alone is not a contradiction)

By combining the two above statements we get that $M'(e)$ can neither converge, nor diverge, which is a contradiction. ∎

This proof may look unmotivated— why define M' as we did? We now look at how one might have come up with the halting set if one's goal was to come up with an explicit set that is not decidable:

We want to come up with a set A that is not decidable. So we want that M_1 does not decide A, M_2 does not decide A, etc. Let's make A and machine M_i differ on their value of i. So we can DEFINE A to be

$$A = \{i \mid M_i(i) \neq 1\}.$$

This set can easily be shown undecidable— for any i, M_i fails to decide it since A and M_i will differ on i. But looking at what makes A hard intuitively, we note that the "$\neq 1$" is a red herring, and the set

$$B = \{i \mid M_i(i) \downarrow\}$$

would do just as well. This is essentially the Halting problem.

Corollary 3.3 *The set $K = \{e \mid M_e(e) \downarrow\}$ is undecidable.*

Proof: In the proof of Theorem 3.2, we actually proved that K is undecidable. ∎

Note 3.4 In some texts, the set we denote as K is called the Halting set. We shall later see that these two sets are identical in computational power, so the one you care to dub THE halting problem is not important. We chose the one we did since it seems like a more natural problem. Henceforth, we will be using K as our main workhorse, as you will see in a later section.

4 Computablely Enumerable Sets

K_0 and K are not decidable. Well, what CAN we say about K_0 that is positive. Lets look back at our feeble attempt to solve K_0. The algorithm was: on input x, y, run $M_x(y)$ until it halts. The problem was that if $\langle x, y\rangle \notin K_0$ then the algorithm diverges. But note that if $(x, y) \in K_0$ then this algorithm converges. SO, this algorithm DOES distinguish K_0 from $\overline{K}_0$. But not quite in the way we'd like. The following definition pins this down

Def 4.1 A set A is *computablely enumerable* (henceforth "r.e.") if there exists a Turing Machine M that behaves as follows:

$$M(x) = \begin{cases} \downarrow & \text{if } x \in A, \\ \uparrow & \text{if } x \notin A. \end{cases}$$

Exercise 1 Show that K and K_0 are r.e.

Exercise 2 Show that if A and B are computable then $A \cap B$, $A \cup B$, and $\overline{A}$ are computable. Which of these are true for r.e. sets?

There is a definition of r.e. that is equivalent to the one given, and is more in the spirit of the words "computablely enumerable."

Theorem 4.2 *Let A be any set. The following are equivalent:*

1. *A is the domain of a partial computable function (i.e. A is r.e.)*
2. *A is the range of a total computable function or $A = \emptyset$ (this definition is more like enumerating a set).*

Proof: We show 1) $\rightarrow$ 2) $\rightarrow$ 1).

1) $\rightarrow$ 2): Let A be the domain of a partial computable function f. Let M be a Turing Machine whose domain is A. If A is empty, then 2) is established. Assume that A is nonempty and let $a \in A$. Let g be the (total) computable function computed by the following algorithm:

1. Input(n).
2. If $n = 0$ then output a.
3. Compute $X = \{g(0), g(1), g(2), \ldots, g(n-1)\}$.
4. Let $Y = \{0, 1, 2, \ldots, n\}$. If $Y - X$ is empty then output a. If $Y - X$ is not empty then run M on every element of $Y - X$ for n steps. If there is some $y \in Y - X$ such that $M(y)$ halts within n steps then output the least such y. Else output a.

We show that range(g) =domain(f). If y is in the range of g then it must be the case that $M(y)$ halted, so y is in the domain of f. If y is in the domain of f then let n be the least number such that $M(y)$ halts in n steps and $y \leq n$. If there is some $m < n$ such that $g(m) = y$ then we are done. Otherwise consider the computation of $g(n)$. In that computation $y \in Y$ but might not be output if there is some smaller element of Y. The same applies to $g(n+1), g(n+2), \ldots$. If there are z elements smaller than y in A then one of $g(n), g(n+1), \ldots, g(n+z)$ must be y.

2) $\rightarrow$ 1). Assume that A is either empty or the range of a total computable function. If A is empty then A is the domain of the partial computable function that always diverges, and we are done. Assume A is the range of a total computable function f. Let g be the partial computable function computed by the following algorithm:

1. Input(n).
2. Compute $f(0), f(1), \ldots$ until (if it happens) you discover that there is an i such that $f(i) = n$. If this happens then halt. (if it does not, then the function will end up diverging, which is okay by us).

We show that an element n is in the range of f iff $g(n)$ halts. If n is in the range of f then there exists an i such that $f(i) = n$; this i will be discovered in the computation of g on n, so $g(n)$ will be 1. If $g(n)$ halts then an i was discovered such that $f(i) = n$, so n is in the range of f.

∎

Several questions arise at this point:

- Are there any sets that are r.e. but not computable?
- Are there any sets that are NOT r.e.?
- If a set is r.e., then is its complement r.e. ?

The second question can be answered in a cheap way: since there are an uncountable number of sets and a countable number of r.e. sets (since there are only a countable number of Turing Machines), there are an uncountable number non-r.e. sets. While this is true, it is not a satisfying answer. We will give more concrete answers to all these questions.

First we relate r.e. and computable sets.

Theorem 4.3 *A set A is computable iff both A and $\overline{A}$ are r.e.*

Proof: If A is computable then $\overline{A}$ is computable. Since any computable set is r.e. both are r.e.

Assume A and $\overline{A}$ are r.e. Let M_a be a Turing Machine that has domain A and M_b be a Turing Machine that has domain $\overline{A}$. The set A is computable via the following algorithm: on input x run both $M_a(x)$ and $M_b(x)$ simultaneously; if $M_a(x)$ halts then output YES, if $M_b(x)$ halts then output NO. Since either $x \in A$ or $x \in \overline{A}$, one of these two events must happen. ∎

This theorem links two of our questions: there exists an r.e. set that is not computable iff r.e. sets are not closed under complementation.

5 Reductions

We want a notion of *if you could compute B then you can compute A* We give two such notions.

Def 5.1 $A \leq_{\mathrm{m}} B$ if there is a computable total f such that $x \in A$ iff $f(x) \in B$.

If $A \leq_{\mathrm{m}} B$ and you somehow had access to B then you could compute A. But not that your computing of A only asks ONE question to B and the answer is the answer for A. We define a reduction where you get to ask B lots of questions and even have what you ask depend on the prior answers.

Def 5.2 (Informal) Let A be any set. A set B is *computable in A* if there is a Turing Machine that, together with a "subroutine" for A can decide B. The set A is called an oracle. We denote the fact that B is computable in A by $B \leq_T A$, and say that "B is Turing-less than A"

It is easy to see that for all sets A, $A \times \overline{A} \leq_{\mathrm{T}} A$.

6 Sets that are even harder than HALT

Are there sets that are even "harder to decide" then HALT? We first say what this means formally:

Def 6.1 If $A \leq_{\mathrm{T}} B$, but $B \not\leq_{\mathrm{T}} A$, then B is *harder than* A.

In this section we exhibit sets that are harder than K but do not prove this.

Recall that K can be written as

$$K = \{e \mid (\exists s) M_e(e) \text{ halts in } s \text{ steps } \}.$$

Note that we have one quantifier followed by a COMPUTABLE statement.

How can TOT be written:

$$TOT = \{e \mid (\forall x)(\exists s) M_e(x) \text{ halts in } s \text{ steps}\}.$$

This is two quantifiers followed by a computable statements.

It turns out that TOT cannot be written with only one quantifier and is harder than K. We can classify sets in terms of how many quantifiers it takes to describe them. Adjacent quantifiers of the same type can always be collapsed into one quantifier.

Def 6.2 Σ_n is the class of all sets A that can be written as

$$A = \{x \mid (\exists y_1)(\forall y_2) \cdots (Q y_n) R(x, y_1, y_2, \ldots, y_n)\},$$

where R is a computable relation and Q is $\exists$ if i is odd, and $\forall$ if i is even.

Def 6.3 Π_n is the class of all sets A that can be written as

$$A = \{x \mid (\forall y_1)(\exists y_2) \cdots (Q y_n) R(x, y_1, y_2, \ldots, y_n)\},$$

where R is a computable relation and Q is $\forall$ is i is odd, and $\exists$ if i is even.

Def 6.4 A set is Σ_n-complete if $A \in \Sigma_n$ and for all sets $B \in \Sigma_n$, $B \leq_{\mathrm{m}} A$.

We now state a theorem without proof.

Theorem 6.5 *For every i there are sets in $\Sigma_i - \Pi_i$, there are sets in $\Sigma_{i+1} - \Sigma_i$, there are Σ_i-complete sets, and there are Π_i-complete sets.*

Exercise 3 (You may use the above Theorem.) Show that a Σ_i-complete set cannot be in Π_i.

Exercise 4 Show that K is Σ_1-complete. Show that $\overline{K}$ is Π_1-complete.

Exercise 5 Show that if A is Π_i-complete then $\overline{A}$ is Σ_i-complete.

We show that FIN (the set of indices of Turing machines with finite domain) is in Σ_2 and that COF (the set of Turing machines with cofinite domains) is in Σ_3. It turns out that FIN is Σ_2-complete, and COF is Σ_3-complete, though we will not prove this. As a general heuristic, whatever you can get a set to be, it will probably be complete there.

$$FIN = \{e \mid (\exists x)(\forall y, s)[\text{ If } y > x \text{ then } M_{e,s}(y) \uparrow\}$$

$$COF = \{e \mid (\exists x)(\forall y)(\exists s)[\text{ If } y > x \text{ then } M_{e,s}(y) \downarrow\}$$