Math 29: Complete Sets

May 6th, 2022

1 Completeness

A set M is **complete** for a class of sets $\{X_i\}_{i\in\omega}$ and a reducibility $\leq$ if $X_i \leq M$ for all i. We've already seen some examples of complete sets: a c.e. set A is 1-complete if $W_e \leq_1 A$ for all e. In other words, all c.e. sets are 1-reducible to A. We proved before the midterm that the 1-complete sets are exactly the creative sets.

Theorem 1. *(The Index Set Theorem) If A is a nontrivial index set, then either $K \leq_1 A$ or $K \leq_1 A^c$.*

Proof: Note that it suffices to prove $K \leq_1 A$ or $K \leq_1 A^c$, as $W_e \leq_1 K$ for all e and $\leq_1$ is transitive.

Let e be an index for the machine which never converges on any input, which is either in A or A^c, and let k be in the opposite. (Which exists since A is nontrivial.) By the s-m-n, there is a total computable injection f such that

$$\varphi_{f(n)}(m) = \begin{cases} \varphi_k(m) & \text{if } \varphi_n(n)\downarrow \\ \uparrow & \text{otherwise} \end{cases}$$

Thus $\varphi_{f(n)} = \varphi_k$ if $n \in K$ and $\varphi_{f(n)} = \varphi_e$ if $n \notin K$. As A, and by extension its complement, are index sets, f 1-reduces K to whichever of A or A^c does not contain e.

Notice that this gives an alternate proof of Rice's theorem, as $A \leq_1 B$ and B being computable implies that A is computable.

In particular, any non-trivial c.e. index set is 1-complete, and hence creative. C.e. index sets exist, but describing exactly which ones are c.e. we will not do. All c.e. sets are still 1-reducible to those which are not c.e., however. Many of these provide interesting examples of problems which the halting problem can be 1-reduced to.

- $Nem = \{e : W_e \neq \emptyset\}$
- $Fin = \{e : W_e \text{ is finite}\}$
- $Inf = \{e : W_e \text{ is infinite}\}$
- $Tot = \{e : W_e = \omega\}$
- $Con = \{e : W_e^c \text{ is finite}\}$
- $Rec = \{e : W_e \text{ is computable}\}$
- $Ext = \{e : W_e \text{ can be extended to a total computable function.}\}$

All of these are nontrivial, so K is 1-reducible to either them or their complements. For some of them, it is 1-reducible to both.

We shall see that not all of the above are at the same level. While the exact definitions necessar to formalize this idea will be covered over the coming weeks by Ben in the X-hours, the number and order of quantifiers it takes to state something relates directly to how hard it is to solve. Notice that Nem only takes a single existential quantifier to state:

$$Nem = \{e : \exists m \ m = \langle n, s\rangle \text{ and } \varphi_{e,s}(n)\downarrow\}$$

Conversely, Tot requires two quantifiers:

$$Tot = \{e : \forall n \ \exists s \ \varphi_{e,s}(n)\downarrow\}$$

It is possible to artificially add quantifiers, but it turns out that there is some minimum number of quantifiers (and specific order/type of them!) which is optimal for a given statement. In the above examples, a single existential quantifier is optimal for Nem and $\forall\exists$ is optimal for Tot. In the X-hours, you will see that quantifiers correspond to the **arithmetical hierarchy**, which are closely related to the **Borel hierarchy** from topology and analysis. We will get a sequence of collections of sets

$$\Delta_1^0, \Delta_2^0, \Delta_3^0, \ldots$$

$$\Sigma_1^0, \Sigma_2^0, \Sigma_3^0 \ldots$$

$$\Pi_1^0, \Pi_2^0, \Pi_3^0 \ldots$$

The above index sets will be complete for some level of this hierarchy. Nem is Σ_1^0 complete (and 1-complete by the Index Set Theorem), Tot is Π_2^0 complete, etc.

However, to be able to fully describe this phenomenon, we first need to define Turing reducibility, which is the "right" notion of reducibility to capture our intuition. To do this, we need the concept of **oracles** and oracle machines.

2 Oracle Machines

An **oracle Turing machine** is a Turing machine which has a second input tape and a second head. The second head can be moved along the second input tape, but it cannot write to it. Instructions now act based on the current cell on both tapes, and valid actions are writing to the first tape, or moving along either tape. Machines are still required to come with finitely many instructions, and we can still use the same principles for coding machines as we did with standard Turing machines.

Alternatively, we can define **oracle register machines**. These contain a countable, one-way input tape containing either 0's or 1's, with cells indexed by natural numbers. For each register R_i, we have a new instruction R_i^o, which reads the cell indexed by the current value contained in R_i and has two output nodes: a 0 node and a 1 node based on the value on the tape. As in the previous case, we can extend our existing coding without issue.

An **oracle** is then just a set whose characteristic function is fed into the oracle tape. For the same reasons as before, there is a universal oracle machine O. It is very important to specify that this is only universal in that it can simulate the run of any oracle machine - it **cannot** simulate all oracles. There are uncountably many oracles, but only countably many oracle machines which the universal machine can run.

We use $\Phi_e(n)$ to denote running e-th oracle machine on input n, i.e. $\Phi_e(n) = O(\langle e, n\rangle)$. Given a set $X \subseteq \omega$, $\Phi_e^X(n)$ denotes the run of the e-th oracle machine on input n with the characteristic function of X on the oracle tape. All of our notation for φ_e carries over - $\Phi_{e,s}^X(n)$ is time bounded computation, $\Phi_e^X(n)\uparrow$ and $\Phi_e^X(n)\downarrow$ mean that the machine diverges or converges respectively, etc.

In terms of the Church-Turing thesis, one should think of oracles as a library function which we could not code ourselves, but which we can nonetheless call. We can ask questions about membership of a number in our oracle and make decisions based on that value. If our oracle is computable, then it is clear that our process is computable: it is just an example of the "black box" nodes we used in our original register machines. However, when X is NOT computable, then there is something interesting to be said. This is the foundation for Turing reducibility, which we will define formally in the next class.