Math 29: Oracles

May 9th, 2022

1 Oracles

Recall that an oracle machine can be thought of as a compute program which has access to a function, *oracle(n)*, which returns 0 or 1 based on whether or not n is in the **oracle**. When we run the oracle machine, we have some $X \subseteq \omega$ as the input oracle, which determines which n return 1 and which return 0.

It is important to note the distinction between an oracle machine, and the run of an oracle machine with a specific oracle. An oracle machine is just a Turing or register machine with an added function, so there are still only countably many machines. They still have an effective coding, i.e. Φ_e represents the e-th oracle machine, and there is a universal oracle machine. Φ_e^X represents feeding the oracle X into the e-th machine.

We mentioned the following fact when we discussed effectively simple sets, which we now proof to illustrate the use of oracles in computation.

Theorem 1. *If S is effectively simple, then there is an oracle machine Φ_e such that $\Phi_e^S = \chi_K$.*

Proof: Recall that a set is effectively simple if it is c.e. and there is total computable function f such that $|W_i| < f(i)$ for all i such that $W_i \subseteq S^c$. Let f be such for S.

As K is c.e., it is equal to W_k for some k. We will consider K in stages, i.e. the sequence of sets $\{K_s\}_{s\in\omega}$ which are the domains of the functions $\varphi_{k,s}$. (I.e., $\{W_{k,s}\}_{s\in\omega}$.) We call this an **enumeration** of K. Similarly, because $S = W_j$ is c.e., there is an enumeration $\{S_s\}_{s\in\omega}$.

Given x, define $\theta(x)$ to be the least s such that $x \in K_s$. In other words, this is the partial computable function which outputs the least stage at which x enters K. As in the canonical simple set construction, let $S_s^c = \{a_0^s < a_1^s < \dots\}$.

Proof: (Cont.) Then by the recursion theorem with parameters, there is $h(x)$ such that

$$W_{h(x)} = \{a_0^{\theta(x)} < a_1^{\theta(x)} < \cdots < a_{f(h(x))}^{\theta(x)}\}$$

if $x \in K$, and $W_{h(x)} = \emptyset$ otherwise. In other words, given x and m, compute $\theta(x)$. If it converges, then enumerate the first $f(m)$ elements of the complement of $S_{\theta(x)}$. Then apply the recursion theorem to get a fixed point for this process in terms of x. So far, everything we have done is completely computable.

Define $r(x)$ to be the least s such that $a^s_{f(h(x))} = a_{f(h(x))}$. In other words, the smallest stage such that the $f(h(x))$-th element of the complement at stage s is the correct $f(h(x))$-th element of the complement in the end. Because c.e. sets never undo enumerated elements, this means that the first $f(h(x))$ elements of the complement are finalized and will not change moving forward, i.e. $a^t_{f(h(x))} = a_{f(h(x))}$ for all $t \geq s$. Furthermore, r is total because of this. Notice that this is not computable in general: we cannot know for sure when the first few elements of the complement have stabilized. **However, if we have access to S as an oracle, then we CAN compute** $r(x)$**.** Therefore, we can compute everything up to this point using S as an oracle.

Now suppose $x \in K$ and $r(x) \leq \theta(x)$. Then this means

$$W_{h(x)} = \{a_0^{\theta(x)} < a_1^{\theta(x)} < \cdots < a_{f(h(x))}^{\theta(x)}\} = \{a_0 < a_1 < \cdots < a_{f(h(x))}\}$$

is a subset of S^c by definition. But this is a contradiction, as $|W_{h(x)}| = f(h(x)) + 1 > f(h(x))$ contradicts the fact that S is effectively simple via f. Therefore, $x \in K$ if and only if $r(x) > \theta(x)$, so $x \in K$ if and only if $x \in K_{r(x)}$. Therefore, the oracle machine with access to S which takes x as input, then calculates $r(x)$ and enumerates $K_{r(x)}$. It then checks if $x \in K_{r(x)}$, which matches whether or not $x \in K$.

This justifies our discussion on why 1-reducibility and m-reducibility were not the correct notion: here, we are able to use information about any effectively simple set to compute membership of the halting problem, but $K \not\leq_1 S$ and $K \not\leq_m S$. In both of these reducibilities, we are only allowed to ask one question of the oracle: we have a total computable (injective) function f, which does most of the work determining which question we should ask of the oracle. As the name suggests, this is similar to ancient Greek pilgrims traveling to Delphi to ask a limited number of questions on which to base their decisions.

Limitations on how many oracle calls we are allowed to use and when lead to

a host of other reducibilities, such as truth table reducibility, weak truth table reducibility, etc. We will not go into detail with these, instead starting with **Turing reducibility** next time.

2 The Finite Use Principle

One facet of oracle computation is worth stressing. Note that, as with Turing computation, runs of an oracle machine are finite when they halt: finitely many bits of the tape have been read or written, finitely many instructions have been run, **and finitely many oracle queries have been asked.** In particular, if we pass in a separate oracles which agree on the same elements which are queried, then the computation will run exactly the same.

To see an illustration of this, observe the proof of the above theorem. To determine if $x \in K$, we need to know the first $f(h(x))$ elements of S^c. If these do not change, then neither will $r(x)$, so the determination of whether x is an element of K will proceed the same regardless of the rest of the set.

Given a natural number n and a subset X of ω, $X \upharpoonright n$ represents "X restricted to n," i.e. the finite binary string of length n whose i-th bit is $\chi_X(i)$. E.g., if E is the set of evens, then $E \upharpoonright 4 = 1010$. If $\sigma \in 2^{<\omega}$ (the set of finite binary strings) and $\sigma(i) = \chi_X(i)$ for all $i < |\sigma|$ (the length of σ), then we say X **extends** σ and write $\sigma \preceq X$.

Given $\sigma \in 2^{<\omega}$, let $\Phi^{\sigma}_{e,s}(n)$ represent running the oracle machine with σ on the oracle tape for s steps, diverging if the machine has not halted by then **or** if the machine attempts to query the oracle for some value not defined by σ. If the s is omitted, then we will let $\Phi^{\sigma}_{e}(n)$ represent the computation $\Phi^{\sigma}_{e,|\sigma|}(n)$. That is, we let the length of the oracle input bound the time the computation is allowed to run.

So, $\Phi_e^{11001100}(n)$ runs $\Phi_e(n)$ for 8 steps, returning the appropriate value if the oracles is queried for numbers less than 8, but diverging if a larger value is queried. We use $\uparrow$ and $\downarrow$ to represent divergence and convergence as per usual.

The **use** of a computation $\Phi^X_e(n)$ which converges is denoted by $\mu^X_e(n)$, and is defined to be the largest k whose membership in X is queried by the run of the computation. (If the computation diverges, then its use does as well.) Then, if $\tau = X \upharpoonright \mu^X_e(n)$ and $\tau \preceq A$,

$$\Phi^X_{e,s}(n) = \Phi^A_{e,s}(n) = \Phi^{\tau}_{e,s}(n)$$

We will use this in priority constructions: if we are building a set X by defining its characteristic function as an increasing sequence of finite binary strings, then so long as we do not go back and change things, any computations we see converge will converge no matter what we do with the rest of X.