Louise Hay Logic Seminar notes

Noah Schoem

February 4, 2016

I am talking today about Turing Degrees and the Jump Operator.

1 Basics

We start with a quick review of basics:

Definition 1. A Turing machine is a machine with a two-way infinitely long tape of *cells* containing 0, 1, or B, a *read/write head*, and a finite collection of *states* Q, one of which is the starting state and another of which is $HALT$.

Its operation happens in discrete time chunks, where at each time the read/write head reads its current cell, and does something based on the current cell and its state: the read/write head can overwrite the current cell with a 0 or 1, and then moves left or right. The machine halts when it reaches the $HALT$ state.

Definition 2. A set $A \subseteq \mathbb{N}$ is recursive/Turing computable if there is a Turing machine that always halts that decides whether $n \in \mathbb{A}$. A function $f : \mathbb{N} \to \mathbb{N}$ is computable if it can be computed with a Turing machine.

Definition 3. A set A is recursively enumerable if there is a Turing machine such that $x \in A$ iff when x is written on the read/write tape in binary, the Turing machine halts and returns a 1.

Theorem 4. *There are countably many Turing machines, and we can code them computably (in the sense that we can write a Turing machine that halts and produces every possible state diagram) as P_e over $n \in \mathbb{N}$; the associated partial function is written Φ_e.*

Theorem 5. *There is a universal Turing machine U that takes as input a pair (n, e) and simulates $P_e(n)$.*

Theorem 6. $K := \{n \in \mathbb{N} | P_n(n) \downarrow\}$ *is r.e. but not recursive.*

2 Relativization

We want to develop some notion of relative computability gague just how hard the questions we ask are. We already have $\leq_1$ and $\leq_m$. These aren't great, because we'd like to be able to say that $\bar{A}$ and A are equicomputable. But $\bar{A} \not\leq_m A$ for some sets A. This motivates a new way of thinking about computation:

Definition 7. A *relativized* Turing machine (*oracle tape* Turing machine) is a Turing machine where there is an added two-way infinite tape. The read/write head moves along the oracle tape simultaneously with the work tape, but may only read it. The oracle tape has written on it the characteristic function of some set A.

Definition 8. Given an oracle tape Turing machine, we write $\hat{P}_e$ to denote the eth oracle tape Turing machine, and given an A, we write Φ_e^A for the corresponding partial function.

We may now talk about Turing reducibility.

Definition 9. A function f is *Turing computable in* A, written $f \leq_T A$, if there is a program $\hat{P}_e$ such that $f(x) = y$ iff $\hat{P}_e(x)$ halts whenever A is on its oracle tape and $\Phi_e^A(x) = y$. $B \subseteq \omega$ is computable in A if $\chi_B \leq_T A$ (and we'll write $B \leq_T A$).

If you're following along in Soare, he writes $\{e\}^A$ to denote Φ_e^A.

It should be clear that $\bar{A} \leq_T A$: If A is on an oracle tape, just have a program that, given n, finds the nth cell on the oracle tape. If the contents of that cell are d, output $1-d$.

The definition above is "relativized", in the sense that we took the definition of Turing machines and made it "relative" to A. This works analogously for the definitions of A-partial recursive, A-recursive, and A-r.e. Many statements and theorems about Turing machines and partial recursive functions relativize as well:

Theorem 10 (Relativized Enumeration Theorem)**.** *There exists a $z \in \omega$ such that for all $A \subseteq \omega$ and all $x, y \in \omega$, $\Phi_z^A(x, y) = \Phi_x^A(y)$.*

This is just the existence of a universal A-relativized Turing machine, and the proof has no remarkable differences from the non-relativized version.

Theorem 11 (Relativized s-m-n theorem; Relativized Parametrization lemma)**.** *For every m, n there is an injective recursive $m+1$-ary function s_n^m such that for all $A \subseteq \omega$ and for all $x, y_1, \ldots, y_n \in \omega$,*

$$\Phi^A_{s_n^m(x,y_1,\ldots,y_n)} = \lambda z_1, \ldots, z_n[\Phi_x^A(y_1, \ldots, y_m, z_1, \ldots, z_n)]$$

There's not much to say about this, other than it works just like the non-relativized version.

(If someone really wants to know:)

Proof. Using a suitable pairing function, we may take $m = n = 1$. Then let $\hat{P}_{s(x,y)}(z) = \hat{P}_x(y, z)$. Since our enumeration of the relativized Turing machines is effective, s is recursive. s can be made injective by the usual padding. $\square$

Theorem 12 (Relativized Recursion Theorem)**.** *For all $A \subseteq \omega$ and all $x, y \in \omega$, if $f(x, y)$ is recursive in A then there is a recursive n such that $\Phi^A_{n(y)} = \Phi^A_{f(n(y),y)}$.*

Furthermore, if $f(x, y) = \{e\}^A(x, y)$ then $n(y)$ can be made uniform in e; that is, n does not depend on A.

Proof. Left as exercise. You read the standard Recursion theorem and make sure relativization doesn't change things. $\square$

The uniformity part of this is notable, since it does not occur in other relativized theorems.

It's worth mentioning that computation with oracles, when a computation halts, does so finitely. This has a name:

Theorem 13 (Use Principle)**.** *If $\Phi_e^A(x) = y$ then there is a finite $\sigma \subseteq A$ and an s such that $\Phi_e^\sigma(x)$ halts in s steps with output y.*

We'll conclude our remarks on relative computability by tying the notion of Turing computability to a syntactic notion:

Definition 14.

1. We say that a set B is r.e. in A if B is the domain of an A-recursive function, i.e. for some e, $B = W_e^A$.
2. We say that a set B is Σ_1^A if there is an A-recursive R such that $B = \{x | \exists y(R^A(x, y))\}$.

Theorem 15. *B is r.e. in A iff B is Σ_1^A.*

Proof. The reverse direction is just the relativized version of the standard proof that B is r.e. iff B is Σ_1.

For the forward direction, by the Use Principle, $x \in B \iff \exists s, \sigma(\sigma \subseteq A \wedge x \in W^\sigma_{e,s})$. Observe that $x \in W^\sigma_{e,s}$ is recursive, and $\sigma \subseteq A$ is A-recursive and equivalent to $\forall y < len(\sigma), \sigma(y) = A(y)$. Thus $B = \{x | \exists s \exists \sigma R(e, x, \sigma, s)\}$ where R is an A-recursive relation on x, s, σ. $\square$

3 Turing degrees

We already saw the relation $\leq_T$, which is reflexive (obviously) and transitive (and to see this, if $A \leq_T B$ and $B \leq_T C$, just make a relativized TM with oracle tape C that whenever we need to check an answer to B, we compute that answer using C). Thus $\equiv_T$ is an equivalence relation.

Definition 16. The equivalence classes of $\equiv_T$ are the Turing degrees, denoted $deg(A)$.

Definition 17. We write $deg(A) \cup deg(B)$ to mean $deg(A \oplus B)$, the degree of their computable join.

Theorem 18. *$deg(A \oplus B)$ is the least upper bound of $deg(A)$ and $deg(B)$.*

Proof. Clearly $A, B \leq_T A \oplus B$ so suppose that $A \leq_T X$ and $B \leq_T X$. Then to compute $A \oplus B$ using X, write a program that on input $2n$, computes $\chi_A(n)$ using X, and on input $2n+1$, computes $\chi_B(n)$ using X. $\square$

Remark 19. The computable join can be extended to a join of ω-many sets $\bigoplus_n A_n$ which is the minimum degree that computes all the A_n's uniformly. We cannot extend to all ordinals, or even all countable ordinals. Computable join, for instance, doesn't make sense to extend to ω_1^{CK} since we want the join to reflect information about the sets we're joining, not the index we're joining them over. We *can*, however, extend the join to *recursive* ordinals, that is, ordinals α such that there is a recursive bijection $f : \omega \to \alpha$. The case of ω is an exercise in Soare.

Definition 20.
1. Given an $A \subseteq \omega$, let $K^A = \{x \in \omega | \Phi_x^A(x) \downarrow\}$. We say that K^A is the "jump" of A, and is often denoted A'.
2. We may iterate this operation to obtain $A^{(n)}$.

Theorem 21.
1. *A' is recursively enumerable in A.*
2. *$A' \not\leq_T A$.*

These should be pretty clear.

As a special case, we say 0 is the degree of all computable sets, and then $0^{(n)}$ is the nth Turing jump of 0. This gives us a hierarchy $0 < 0' < \cdots < 0^{(n)} < \ldots$.