

Chapter 6

Degrees of Unsolvability

Relative computability

All of the non-computable sets are non-computable, but some are more non-computable than others. We want to make sense of this idea.

For example, suppose A and B are both non-computable subsets of \mathbb{N} . On the one hand, we might be able to show that if, hypothetically speaking, we could somehow decide membership in B , then we could decide membership in A . This would lead us to the opinion that A was no more undecidable than B was.

On the other hand, we might be able to show that even if our Fairy Godmother gave us an oracle so we could decide membership in B , there *still* would be no decision procedure for A . This might lead us to the opinion that A was *more* undecidable than B , or else that their “degrees of unsolvability” were not directly comparable.

Now an “oracle” for B sounds like a magic device—a black box sitting on a tripod, perhaps, with the ability to say whether or not a given number belongs to B . The informal concept we want to explore is the concept of effective calculability *relative to* some fixed set B . Our mental image of effective calculability (in Chapter 1) involved a clerk dutifully carrying out given instructions. Now we intend to supply this clerk with one more asset: an oracle for B . This will allow him to do more than before (if B is undecidable); he can now very easily calculate the characteristic function C_B of B , for example. But there will still be limits to what he can do, even with this new oracle.

The remarkable fact is that we can turn this into mathematics, without any magic. The concept of relative computability first appeared in a 1939 paper by Alan Turing. At first glance, it might seem strange to combine the rather constructive concept of computability with the almost mystical idea of an oracle. It is to Turing’s credit that he perceived that the combination, strange or not, would be a useful tool in classifying the non-computable sets.

The preceding paragraphs give an informal description of the concept of effective calculability relative to a set B (shades of Chapter 1!). Here is the plan. We want to make the concept into a genuine mathematical concept in two ways: general recursiveness relative to B (as in Chapter 2), and register-machine computability relative to B (as in Chapter 3). So next we will look at the (rather minor) changes needed to Chapter 2 to incorporate B . And then we will look at the changes needed to Chapter 3 to incorporate B . In particular, we will want to know that two approaches yield exactly the same class of partial functions. And we will want to know that the theorems from Chapters 2 and

⁰Chapters 5, 6, and 7 are independent, and can be read in any order.

3 continue to hold in the new setting. After that, we will explore what use can be made of these ideas.

In Chapter 2, we formalized the concept of effective calculability by defining the class of general recursive partial functions: the partial functions that could be built up from certain initial functions by use of composition, primitive recursion, and search. Define the *general recursive partial functions relative to B* in exactly the same way, except for allowing one additional initial function, namely the characteristic function C_B of B .

Moreover, we obtain the *primitive recursive functions relative to B* by foregoing unbounded search. And an n -ary relation R on the natural numbers is defined to be *primitive recursive relative to B* if its characteristic function C_R is primitive recursive relative to B . Similarly, a relation R is *general recursive relative to B* if its characteristic function C_R (which is always total) is general recursive relative to B .

For a simple example, the complement \overline{B} of B is primitive recursive relative to B , because its characteristic function is obtainable by composition from the function $x \mapsto 1 \dot{-} x$ and C_B .

As in Chapter 2 (and by the same proofs), the class of relations that are general recursive relative to B is closed under the constructions we came across there. In particular, it is closed under unions, intersections, complements, bounded quantifiers, and substitution of total functions that are general recursive relative to B .

(It is clear that this concept of relative general recursiveness could be extended in two ways. First, instead of adding the C_B as an initial function for *one* set B , we could add the characteristic functions of *many* sets. That is, where \mathcal{C} is some collection of subsets of \mathbb{N} , we can define the class of general recursive partial functions relative to \mathcal{C} by including all the characteristic functions of members of \mathcal{C} as initial functions. And here \mathcal{C} could be finite or infinite.

Secondly, for a total function F on the natural numbers, we can obtain the general recursive partial function relative to F by adding F as an initial function. That is, there is no necessity of limiting ourselves to oracles for characteristic functions—we can handle oracles for other total functions just as well.

Combining these two extensions, where \mathcal{F} is a collection of total functions, we can obtain the general recursive partial functions relative to \mathcal{F} by adding all of \mathcal{F} as initial functions.

These are good extensions, but we will not make use of them right now. For the time being, we will concentrate solely on the concept of computability relative to a single set B of natural numbers.)

In Chapter 3, we examined the register machine option for formalizing the concept of effective calculability. Now we add a new type of instruction for register machines:

- $C\ r$ (where r is a numeral for a natural number). “Convert r .” The effect of this instruction is to replace the number x in register r by $C_B(x)$. This

happens instantly, in one step. The machine then proceeds to the next instruction in the program (if any).

(In the case of computability relative to a total function F , this instruction replaces the x in register r by $F(x)$.)

For a program \mathcal{P} in the new sense, when we begin execution with \vec{x} in registers $1, \dots, n$, there are three possibilities:

- (i) The calculation might never halt.
- (ii) The calculation might come to “good” halt, by seeking the first non-existent instruction.
- (iii) The calculation might come to a “bad” halt, by seeking some other non-existent instruction. (This can happen if the program tries to jump too far backwards or forwards; it can also happen if the last instruction is a decrement instruction.)

As in Chapter 2, we say, for an n -place partial function f , that \mathcal{P} *computes f relative to B* if when we start execution with an n -tuple \vec{x} in registers $1, \dots, n$ and with 0 in all other register, then the following conditions hold:

- If $f(\vec{x})$ is defined (i.e., if $\vec{x} \in \text{dom } f$), then the computation eventually terminates with $f(\vec{x})$ in register 0. Furthermore, the computation comes to a good halt, by seeking a $(p + 1)$ st instruction, where p is the length of \mathcal{P} .
- If $f(\vec{x})$ is undefined (i.e., if $\vec{x} \notin \text{dom } f$), then the computation never terminates.

Then we say that f is a partial function that is *register-machine computable relative to B* if there exists a program \mathcal{P} (which may contain conversion instructions) that computes f relative to B (in the above sense of the verb “computes”).

For example, the characteristic function C_B is register-machine computable relative to B : We use the program that converts register 1, and then moves the result to register 0. We can do this in four or five lines. (And those lines do not depend on what the set B is.)

All of the other initial functions are register-machine computable relative to B , even without using conversion instructions. And the proofs from Chapter 3 show that the class of partial functions register-machine computable relative to B is closed under composition, primitive recursion, and search. No changes are required in any of those constructions. Therefore we have the following result:

Theorem: Every partial function that is general recursive relative to B is also register-machine computable relative to B .

We can code programs as in Chapter 3, assigning Gödel numbers to conversion instructions:

$$\#C r = [4, r]$$

There is no need to change the “loc” function that updates the location counter in executing a program. In the case of a conversion instruction, we already have $\text{loc}(k, m, e) = k + 1$ by the default clause.

The “mem” function for updating the memory number needs to be adapted by adding additional cases for the conversion instruction:

$$\text{mem}_B(m, c) = \begin{cases} m \cdot p_{(c)_1} & \text{if } (c)_0 = 0 \text{ and } c \neq 0 & \text{(increment)} \\ \lfloor m/p_{(c)_1} \rfloor & \text{if } (c)_0 = 1 \text{ and } p_{(c)_1} \mid m & \text{(decrement)} \\ \lfloor m/p_r^x \rfloor & \text{if } (c)_0 = 4 \text{ and } x \notin B & \text{(convert)} \\ \lfloor m/p_r^x \rfloor \cdot p_r & \text{if } (c)_0 = 4 \text{ and } x \in B & \text{(convert)} \\ m & \text{otherwise.} \end{cases}$$

where $r = (c)_1$ and $x = (m)_r^*$. In this equation, we have used only parts that are primitive recursive relative to B , so the function mem_B is primitive recursive relative to B . (Moreover, the construction tree for mem_B does not depend on what B is.) Hence the function is register-machine computable relative to B . (And the program that computes mem_B relative to B does not depend on what B is.)

Our universal program, with mem replaced by mem_B , computes (relative to B) an $(n + 1)$ -place partial function $\Phi_B^{(n)}$ with the property that whenever e is the Gödel number of a program that computes (relative to B) an n -place partial function f , then

$$\Phi_B^{(n)}(e, \vec{x}) = f(\vec{x})$$

for all \vec{x} (where equality of partial functions has the usual meaning). We define

$$\llbracket e \rrbracket_B^{(n)}(\vec{x}) = \Phi_B^{(n)}(e, \vec{x}).$$

Enumeration theorem:

- (i) $\Phi_B^{(n)}$ is an $(n + 1)$ -place partial function that is register-machine computable relative to B .
- (ii) For each number e , the n -place partial function $\llbracket e \rrbracket_B^{(n)}$ is register-machine computable relative to B .
- (iii) Each n -place partial function that is register-machine computable relative to B is $\llbracket e \rrbracket_B^{(n)}$ for some number e .

Thus

$$\llbracket 0 \rrbracket_B^{(n)}, \llbracket 1 \rrbracket_B^{(n)}, \llbracket 2 \rrbracket_B^{(n)}, \dots$$

is a complete list (with repetitions) of all the n -place partial functions that are register-machine computable relative to B .

We obtain a snapshot function $\text{snap}_B^{(n)}$ by using mem_B in place of mem . This function is primitive recursive relative to B . And the following relation $T_B^{(n)}$ is also primitive recursive relative to B :

$$\{\langle u, \vec{x}, t \rangle \mid \llbracket u \rrbracket_B^{(n)}(\vec{x}) \downarrow \text{ in } \leq t \text{ steps} \} = \{\langle u, \vec{x}, t \rangle \mid (\text{snap}_B^{(n)}(u, \vec{x}, t))_0 \geq \text{lh } u \}.$$

Normal form theorem: For any \vec{x} , e , and n ,

$$\llbracket e \rrbracket_B^{(n)}(\vec{x}) = \Phi_B^{(n)}(e, \vec{x}) = ((\text{snap}_B^{(n)}(e, \vec{x}, \mu t[(\text{snap}_B^{(n)}(e, \vec{x}, t))_0 \geq \text{lh } e]))_1)_0^*.$$

Looking at the right-hand side in this equation, we see that every piece defines a partial function that is general recursive relative to B . This proves that every partial function $\llbracket e \rrbracket_B^{(n)}$ that is register-machine computable relative to B is also general recursive relative to B . Combining this fact with its converse proved earlier, we have the following reassuring result.

Theorem: A partial function is general recursive relative to B if and only if it is register-machine computable relative to B .

Church's thesis, extended to the context of relativized computability, is the claim that not only are the two concepts in this theorem equivalent to each other, but the concepts are the *correct* concepts to formalize our ideas of effective calculability, given an oracle that can decide membership in a set B .

As in Chapter 4, we adopt a unified terminology. A partial function meeting the conditions of this theorem will be said to be a partial function *computable relative to B* (or a partial function *recursive relative to B*).

A relation on the natural numbers will be said to be *computable relative to B* (or *recursive relative to B*) if its characteristic function (which is always total) is computable relative to B .

For an extreme case, suppose $B = \emptyset$. A partial function is computable relative to \emptyset if and only if it is a computable partial function (in the unrelativized sense). This holds because the characteristic function of \emptyset is a constantly 0 function. This function is already included as an initial function for the construction of the general recursive partial functions. (An oracle for \emptyset is not a useful device.)

Because the class of functions computable relative to B is closed under composition, we have the following:

Substitution rule: If Q is an n -ary relation computable relative to B , and g_1, \dots, g_n are k -place *total* functions that are computable relative to B , then the k -ary relation

$$\{\vec{x} \mid \langle g_1(\vec{x}), \dots, g_n(\vec{x}) \rangle \in Q\}$$

is computable relative to B .

This concludes our review of Chapters 2 and 3, modified to allow for an "oracle" for a set B . Now we want to make use of that material and see what we can do with it.

A key concept is that of Turing reducibility. For a subset A of \mathbb{N} (that is, a unary relation), we say that A is *Turing reducible* to B (written $A \leq_T B$) if the characteristic function of A is computable relative to B . Informally, saying that $A \leq_T B$ implies that membership in A is no harder to decide than is membership in B . That is, we could decide membership in A if we had an oracle for B . Of course, if A is a computable set, then automatically it is computable relative to anything one wants:

$$A \text{ computable} \implies \text{for any } B, A \leq_T B$$

For example, suppose $A \leq_m B$ under the total computable function f :

$$x \in A \iff f(x) \in B$$

Then $C_A(x) = C_B(f(x))$ and consequently $A \leq_T B$. (Right? Method 2 is to apply composition to C_B and f . Method 3 is to follow a program for f by a conversion instruction.) That is,

$$A \leq_m B \implies A \leq_T B.$$

The converse does not hold, in general. For example, we know that $\overline{K} \not\leq_m K$, because \overline{K} is not r.e. On the other hand, $\overline{K} \leq_T K$. In fact, $\overline{A} \leq_T A$ for any set A . (Right?)

Transitivity lemma: If f is a partial function computable relative to B and $B \leq_T C$, then f is a partial function computable relative to C .

Proof: We are given that B is general recursive relative to C , so there is construction tree \mathcal{T} showing how the characteristic function of B is built up from initial functions (possibly including the characteristic function of C).

Similarly, we are given that there is a construction tree showing how the partial function f is built up. But this tree may have among its leaves the characteristic function of B .

Use grafting. In the latter tree, whenever a leaf (at the bottom of the tree) has the characteristic function of B as an initial function, we graft in the tree \mathcal{T} . The result is a tree that still builds up the partial function f , but now its initial functions are only zero, successor, projection, and the characteristic function of C . Hence f is general recursive relative to C . \dashv

Corollary: If f is a partial function computable relative to B and B is a computable set, then f is a computable partial function (absolutely).

Proof: In the transitivity lemma, take $C = \emptyset$. \dashv

It is clear that $A \leq_T A$ for any set A ; one says that the \leq_T relation is *reflexive* on the collection $\mathcal{P}\mathbb{N}$ of sets of natural numbers. The \leq_T relation also has the following property, which is called *transitivity*.

Proposition: Whenever $A \leq_T B$ and $B \leq_T C$, then $A \leq_T C$.

Proof: In the transitivity lemma, take f to be the characteristic function of A . \dashv

Proposition: Whenever $A \leq_T B$ and B is a computable set, then A is also a computable set.

Proof: In the previous proposition, take $C = \emptyset$. \dashv

This proposition, in the contrapositive, has the following consequence: One strategy for showing that a set B is *not* computable is to show that $A \leq_T B$ where A is some set already known to be not computable.

Relations (such as \leq_T or \leq_m) that are reflexive and transitive are called *preorderings*. We will examine the properties of preorderings shortly.

We say that sets A and B are *Turing equivalent* (written $A \equiv_T B$) if each is Turing reducible to the other:

$$A \equiv_T B \iff A \leq_T B \text{ and } B \leq_T A.$$

Because \leq_T is reflexive on $\mathcal{P}\mathbb{N}$, it follows that the \equiv_T relation is also reflexive on $\mathcal{P}\mathbb{N}$, that is, that $A \equiv_T A$ for every set A . Moreover, the transitivity of \leq_T tells us that \equiv_T is also transitive:

$$A \equiv_T B \text{ and } B \equiv_T C \implies A \equiv_T C$$

In addition, \equiv_T has a property that \leq_T lacks: it is *symmetric*:

$$\text{Whenever } A \equiv_T B \text{ then } B \equiv_T A.$$

In summary, the relation \equiv_T is reflexive on $\mathcal{P}\mathbb{N}$, transitive, and symmetric. Such relations are called *equivalence relations* on $\mathcal{P}\mathbb{N}$.

Equivalence relations

Suppose that E is a binary relation on some set U (that is, $E \subseteq U \times U$). In place of

$$\langle x, y \rangle \in E$$

we will write simply xEy .

Definition: (i) E is said to be *reflexive on U* if xEx for every x in U .

(ii) E is said to be *symmetric* if whenever xEy then also yEx .

(iii) E is said to be *transitive* if whenever both xEy and yEz then also xEz .

(iv) E is said to be an *equivalence relation on U* if it is reflexive on U , symmetric, and transitive.

Example 1: The Turing equivalence relation \equiv_T is an equivalence relation on $\mathcal{P}\mathbb{N}$.

Example 2: The relation \leq_m of many-one equivalence is reflexive on $\mathcal{P}\mathbb{N}$ (that is, $A \leq_m A$) and is transitive. Define A and B to be many-one equivalent (written $A \equiv_m B$) if each is many-one reducible to the other:

$$A \equiv_m B \iff A \leq_m B \text{ and } B \leq_m A$$

This relation is also an equivalence relation on $\mathcal{P}\mathbb{N}$.

Example 3: For sets A and B , say that A is *one-one reducible to B* (written $A \leq_1 B$) if $A \leq_m B$ under a function that is (in addition to being total and computable) one-to-one. This relation is also reflexive on $\mathcal{P}\mathbb{N}$ because $A \leq_1 A$ under the identity function. And it is transitive. Make the inevitable definition:

$$A \equiv_1 B \iff A \leq_1 B \text{ and } B \leq_1 A$$

This relation is also an equivalence relation on $\mathcal{P}\mathbb{N}$.

Example 4: Equivalence relations arise throughout mathematics, especially in algebra. For integers x and y , define $x \equiv y$ to hold if the difference $|x - y|$ is evenly divisible by 6. Then \equiv is an equivalence relation on the set \mathbb{Z} of integers. If $x \equiv y$, then we say that x and y are *congruent modulo 6*.

Definition: For an equivalence relation E on a set U , and an element x in U , define the *equivalence class* $[x]_E$ of x to be the set of all objects t that x is related to:

$$[x]_E = \{t \mid xEt\}.$$

When the relation E is fixed by the context, we can write simply $[x]$.

Example 4 revisited: For congruence modulo 6 on the integers,

$$[2] = \{\dots, -4, 2, 8, 12, \dots\}.$$

Example 1 revisited: For Turing equivalence on $\mathcal{P}\mathbb{N}$, the equivalence class $[\emptyset]$ of the empty set is

$$\{A \subseteq \mathbb{N} \mid \emptyset \equiv_T A\}.$$

This is exactly the collection of computable sets. On the one hand, we have $\emptyset \leq_T A$ for *any* set A , because \emptyset is a computable set (absolutely). On the other hand, whenever $A \leq_T \emptyset$, then A is computable, by a recent proposition.

By contrast, $[K]$ does not contain any computable sets, because whenever $K \leq_T A$ then A cannot be computable (by the same proposition). Thus $[\emptyset]$ and $[K]$ are disjoint; they have no members in common.

Lemma: Assume that E is an equivalence relation on U and that x and y belong to U . Then

$$[x]_E = [y]_E \quad \text{if and only if} \quad xEy.$$

Proof: (\Rightarrow) Assume that $[x]_E = [y]_E$. We know that $y \in [y]_E$ (because yEy), and consequently $y \in [x]_E$ (because $[x]_E$ and $[y]_E$ are the same set). By the definition of $[x]_E$, this means that xEy .

(\Leftarrow) Next assume that xEy . Then

$$\begin{aligned} t \in [y]_E &\implies yEt \\ &\implies xEt \quad \text{because } xEy \text{ and } E \text{ is transitive} \\ &\implies t \in [x]_E. \end{aligned}$$

Thus $[y]_E \subseteq [x]_E$. Since E is symmetric, we also have yEx and we can reverse x and y in the above argument to obtain $[x]_E \subseteq [y]_E$. \dashv

Definition: A *partition* Π of a set U is a collection on nonempty subsets of U that is disjoint and exhaustive, i.e.,

- (a) no two different sets in Π have any common elements, and
- (b) each element of U is in some set in Π .

Theorem: Assume that E is an equivalence relation on U . Then the collection

$$\{[x]_E \mid x \in U\}$$

of all equivalence classes is a partition of U .

Proof: Each equivalence class $[x]_E$ is nonempty (because $x \in [x]_E$) and is a subset of U (because $E \subseteq U \times U$). The main thing that we must prove is that the collection of equivalence classes is disjoint, i.e., part (a) of the definition is met. So suppose that $[x]_E$ and $[y]_E$ have a common element t . Thus

$$xEt \text{ and } yEt.$$

But then xEy and by the lemma, $[x]_E = [y]_E$. \dashv

For an equivalence relation E on a set U , we can define the *quotient set*

$$U/E = \{[x]_E \mid x \in U\}$$

whose members are the equivalence classes. (The expression U/E is read “ U modulo E .”) This is a set of sets. We have the *natural map* (or *canonical map*) $\varphi : U \rightarrow U/E$ defined by

$$\varphi(x) = [x]_E$$

for x in U .

Example 4 revisited: For congruence modulo 6 on the integers the quotient set \mathbb{Z}/\equiv consists of exactly six sets:

$$\mathbb{Z}/\equiv = \{[0], [1], [2], [3], [4], [5]\}$$

Example 1 revisited: For Turing equivalence on $\mathcal{P}\mathbb{N}$, the equivalence classes are called *degrees of unsolvability*, or *Turing degrees*, or simply *degrees*. For example, one of the degrees is $[\emptyset]$, the collection of computable sets.

The concepts of countable and uncountable sets (see the appendix) can be usefully applied here. The set of all possible register-machine programs is countable. One way to see this is to note that the map from programs to their indices is one-to-one. So we have a one-to-one function from the set of all programs into \mathbb{N} .

Proposition: For any fixed set B , the set

$$\mathcal{L}_B = \{A \mid A \leq_T B\}$$

is countable.

Proof: We can map each A in this set to the smallest number that is the Gödel number of a program computing A relative to B . This gives a one-to-one map from \mathcal{L}_B into \mathbb{N} . \dashv

Corollary: Each Turing degree is a countable collection of sets.

Proof: $[B]$ is a subset of \mathcal{L}_B . \dashv

Proposition: The set \mathcal{D} of all Turing degrees is uncountable.

Proof: We use the fact that the union of countably many countable sets is countable. This implies that the union of countably many degrees must be a countable subset of $\mathcal{P}\mathbb{N}$. But the union of *all* degrees is $\mathcal{P}\mathbb{N}$, which is uncountable. \dashv

Preordering relations

Suppose that R is any binary relation on a set U (that is, $R \subseteq U \times U$). As before, we can write xRy to mean that $\langle x, y \rangle \in R$.

Definition: R is a *preordering* on U if it is reflexive on U and is transitive.

Example 1: Turing reducibility \leq_T is a preordering on $\mathcal{P}\mathbb{N}$.

Example 2: Many-one reducibility \leq_m is a preordering on $\mathcal{P}\mathbb{N}$.

Example 3: One-one reducibility \leq_1 is a preordering on $\mathcal{P}\mathbb{N}$.

Example 4': Let U be the set of nonzero integers (positive or negative), and that

$$mRn \iff m \text{ divides } n.$$

Then R is obviously reflexive on U , and transitivity is easy to check.

We want to establish that a preordering relation R on U (a) determines an equivalence relation E on U , and (b) partially orders the set U/E of equivalence classes.

We obtain the equivalence relation E from the preordering R by making it symmetric. Define relation E on U by the condition:

$$xEy \iff xRy \ \& \ yRx$$

for x and y in U . (In other words, the relation E is $R \cap R^{-1}$.)

Proposition: The relation E is an equivalence relation on U .

Proof: The definition makes it clear that E is symmetric. It inherits reflexivity and transitivity from R : For x in U , we have xEy because xRx . If xEy and yEz then we have four facts: xRy , yRx , yRz , and zRy . Regrouping these and using the transitivity of R , we get xRz and zRx , whence xEz . \dashv

Hence the relation E partitions U into equivalence classes. Let $[x]$ denote the equivalence class to which x belongs. We know that $[x] = [y]$ if and only

if xEy . Now consider the quotient set U/E of all equivalence classes. We can define a binary relation \leq on U/E by the condition:

$$[x] \leq [y] \iff xRy$$

for x and y in U .

Caution: There is something to prove here, namely that \leq is “well defined” or “invariant.” Suppose that \mathbf{a} and \mathbf{b} are two equivalence classes. We are attempting to define whether or not $\mathbf{a} \leq \mathbf{b}$ holds by *choosing* a particular x from \mathbf{a} and a y from \mathbf{b} , and then testing to see if xRy . We need to verify that the verdict is independent of the particular choices made. Suppose that instead of x and y , we had chosen $x' \in \mathbf{a}$ and $y' \in \mathbf{b}$. What must be shown is that $xRy \iff x'Ry'$.

Once we see what must be shown, actually showing it is easy. Since x and x' are in the same equivalence class, we have xEx' . Similarly yEy' . It follows from transitivity that

$$xEx' \ \& \ yEy' \ \& \ xRy \implies x'Ry'.$$

Proposition: The relation \leq is reflexive on U/E , transitive, and antisymmetric.

“Antisymmetric” means that whenever both $\mathbf{a} \leq \mathbf{b}$ and $\mathbf{b} \leq \mathbf{a}$ then $\mathbf{a} = \mathbf{b}$. A relation that is reflexive on U/E , transitive, and antisymmetric is called a *partial ordering* on U/E .

Proof: Reflexivity and transitivity are inherited from R . Suppose that both $[x] \leq [y]$ and $[y] \leq [x]$. Then we have both xRy and yRx , whence xEy . Therefore $[x] = [y]$. \dashv

For equivalence classes \mathbf{a} and \mathbf{b} , we write $\mathbf{a} < \mathbf{b}$ to mean that $\mathbf{a} \leq \mathbf{b}$ and $\mathbf{a} \neq \mathbf{b}$.

Proposition: The relation $<$ is irreflexive (i.e., never $\mathbf{a} < \mathbf{a}$) and transitive.

Proof: Irreflexivity is clear. Suppose that $\mathbf{a} < \mathbf{b} < \mathbf{c}$. Then clearly $\mathbf{a} \leq \mathbf{c}$, but could we have $\mathbf{a} = \mathbf{c}$? No, that would imply $\mathbf{a} \leq \mathbf{b} \leq \mathbf{a}$ whence $\mathbf{a} = \mathbf{b}$ by antisymmetry. \dashv

Example 4': In Example 4' above, we have mEn if and only if $|m| = |n|$. Each equivalence class contains exactly two numbers; $[n] = \{n, -n\}$. Under the partial order, $[1]$ is the *least* class, that is, $[1] \leq [n]$ for every n . The partial order is not a total order; for example, $[2]$ and $[3]$ are incomparable. (That means that neither $[2] \leq [3]$ nor $[3] \leq [2]$.) For any class, there is a strictly larger one.

Example 1: From the preordering \leq_T on $\mathcal{P}\mathbb{N}$ we obtain the equivalence relation E which is nothing but the Turing equivalence relation \equiv_T . The equivalence classes are the Turing degrees, and \leq_T determines a partial ordering on the Turing degrees:

$$[A] \leq [B] \iff A \leq_T B$$

We now want to examine in more detail this partial ordering of the Turing degrees.

Ordering degrees

The degree $[\emptyset]$ (call this degree $\mathbf{0}$) consisting of the computable sets is the *least* degree in this partial ordering. That is, for any degree \mathbf{a} , we have $\mathbf{0} \leq \mathbf{a}$ because $\emptyset \leq_T A$ for any set A .

Let $\mathbf{0}'$ be the degree of K . Then $\mathbf{0} < \mathbf{0}'$.

Define a degree to be *recursively enumerable* if it contains an r.e. set. The degree $\mathbf{0}$ is an recursively enumerable degree; in fact *all* of its members, being computable, are r.e. And the degree $\mathbf{0}'$ is an recursively enumerable degree because it contains K . (It also contains sets that are *not* recursively enumerable, such as \overline{K} .)

Proposition: $\mathbf{0}'$ is the largest recursively enumerable degree. That is, $\mathbf{a} \leq \mathbf{0}'$ for any recursively enumerable degree \mathbf{a} .

Proof: Take an r.e. set A in \mathbf{a} . We saw in Chapter 4 that $A \leq_m K$ because K is a complete r.e. set. Therefore $A \leq_T K$. So $\mathbf{a} \leq \mathbf{0}'$. \dashv

In 1944, Emil Post raised the question whether there were any r.e. degrees other than $\mathbf{0}$ and $\mathbf{0}'$. This question, which became known as “Post’s problem,” was finally answered in 1956 (two years after Post’s death), independently by Richard Friedberg (in his Harvard senior thesis) and by Albert A. Muchnik in the Soviet Union. They showed that intermediate r.e. degrees do indeed exist, and in great profusion. (Nonetheless, there seem to be no “natural” examples of such degrees.)

There is no largest degree. We will see shortly how to construct, for each degree \mathbf{a} , a strictly larger degree \mathbf{a}' .

Exercises

1. Define $A \leq_* B$ to be like $A \leq_m B$ except not requiring that the function be computable. That is, $A \leq_* B$ if there exists some total function f , computable or not, for which

$$x \in A \iff f(x) \in B$$

for all x . Then \leq_* is a preordering on $\mathcal{P}\mathbb{N}$. So it determines an equivalence relation \equiv_* and a partial ordering on the quotient set. How many equivalence classes are there, and how are they ordered?

2. Let U be the collection of computable subsets of \mathbb{N} . Then \leq_m (restricted to U) is a preordering on U . So it determines an equivalence relation \equiv_m on U and a partial ordering on the quotient set. How many equivalence classes are there, and how are they ordered?

3. Give an example of a set in the degree $\mathbf{0}'$ that is neither recursively enumerable nor the complement of an recursively enumerable set.

4. Assume that A and B are sets for which $A \equiv_m B$. Show that we also have $A \equiv_T B$. (One says that the equivalence relation \equiv_m *refines* the equivalence relation \equiv_T .)

5. Show that for any partial function f , there is some set B such that f a partial function computable relative to B .

Structure of the degrees

Consider any two sets, say A (of degree \mathbf{a}) and B (of degree \mathbf{b}). Then there are the four disjoint possibilities:

- $\mathbf{a} = \mathbf{b}$. This happens when $A \equiv_T B$. We can think of A and B as being, in some sense, “equally complex,” or as having the same “information content.”
- $\mathbf{a} < \mathbf{b}$. This happens when $A \leq_T B$ but $A \not\equiv_T B$. We can think of A as being, in some sense, “simpler” than B is, or of having less information content than B has.
- $\mathbf{b} < \mathbf{a}$. We can think of B as being, in some sense, “simpler” than A is.
- None of the above. That is, \mathbf{a} and \mathbf{b} are *incomparable*. This happens when $A \not\leq_T B$ and $B \not\leq_T A$. Incomparable degrees do indeed exist.

Theorem: Any two degrees have a least upper bound. That is, for any degrees \mathbf{a} and \mathbf{b} , we can find a degree \mathbf{c} with the two properties

- (i) $\mathbf{a} \leq \mathbf{c}$ and $\mathbf{b} \leq \mathbf{c}$, and
- (ii) whenever \mathbf{d} is a degree for which both $\mathbf{a} \leq \mathbf{d}$ and $\mathbf{b} \leq \mathbf{d}$, then $\mathbf{c} \leq \mathbf{d}$.

If either $\mathbf{a} \leq \mathbf{b}$ or $\mathbf{b} \leq \mathbf{a}$, this theorem is trivial; we simply take \mathbf{c} to be the larger degree. The theorem is interesting only in the case where \mathbf{a} and \mathbf{b} are incomparable.

Proof: Choose a set A from degree \mathbf{a} and a set B from degree \mathbf{b} . Define the set

$$A \oplus B = \{2x \mid x \in A\} \cup \{2x + 1 \mid x \in B\}.$$

This set codes A on the even numbers, and codes B on the odds. Let \mathbf{c} be the degree of $A \oplus B$.

Then $\mathbf{a} \leq \mathbf{c}$ because $A \leq_1 A \oplus B$ under the function $x \mapsto 2x$. Similarly, $\mathbf{b} \leq \mathbf{c}$ because $B \leq_1 A \oplus B$ under the function $x \mapsto 2x + 1$.

Now suppose that \mathbf{d} is a degree with both $\mathbf{a} \leq \mathbf{d}$ and $\mathbf{b} \leq \mathbf{d}$. Choose a set D from the degree \mathbf{d} . Then $A \leq_T D$ and $B \leq_T D$. We seek to show that $A \oplus B \leq_T D$. The following program computes the characteristic function of

$A \oplus B$ relative to D :

Calculate	parity from 1 to 0, preserving 2	
Calculate	$\lfloor x/2 \rfloor$ from 1 to 1, preserving 1	
D	0	Test parity.
J	*	Jump on even x .
Calculate	C_B from 1 to 0, relative to D	
J	#	Halt.
Calculate	C_A from 1 to 0, relative to D	
		Halt.

Thus $\mathbf{c} \leq \mathbf{d}$. \dashv

A set with a partial ordering is called a *lattice* if any two elements always have a least upper bound and a greatest lower bound. The degrees do *not* form a lattice, because greatest lower bounds do not always exist. The best we can say is that we have an “upper semi-lattice” because we can at least take least upper bounds.

Proposition: For any set B of natural numbers, the collection

$$\mathcal{U}_B = \{C \mid B \leq_T C\}$$

is uncountable.

Proof: For any A in \mathcal{PN} , the set $A \oplus B$ is in \mathcal{U}_B . The map $A \mapsto A \oplus B$ is one-to-one, so \mathcal{PN} has the same size as a subset of \mathcal{U}_B . \dashv

Corollary: For any degree \mathbf{b} , the set $\{\mathbf{c} \mid \mathbf{b} \leq \mathbf{c}\}$ of larger degrees is uncountable.

Proof: Look at the union of this set.

$$\bigcup \{\mathbf{c} \mid \mathbf{b} \leq \mathbf{c}\} = \mathcal{U}_B.$$

On the one hand, the preceding proposition tells us that \mathcal{U}_B is uncountable. On the other hand, the union of countably many countable sets is countable. Hence the union of any countable collection of degrees must be countable. \dashv

A much weaker statement is that the set $\{\mathbf{c} \mid \mathbf{b} \leq \mathbf{c}\}$ (sometimes called the “cone above \mathbf{b} ”) contains more than one degree. So there can be no largest degree; for any degree \mathbf{b} , there are many degrees in the cone above it.

Contrast this result with the following.

Proposition: For any degree \mathbf{b} , the set $\{\mathbf{a} \mid \mathbf{a} \leq \mathbf{b}\}$ of smaller degrees is countable.

Proof: Fix some B in \mathbf{b} . Map each smaller degree \mathbf{a} to the smallest number that is the Gödel number of a register-machine program that computes relative to B some set of degree \mathbf{a} . (We have seen this argument before, back on page

609.) \dashv

Parameter theorem: For each m and n , there is a $(n+1)$ -place primitive recursive function ρ_{mn} such that the equation

$$\llbracket e \rrbracket_B^{(m+n)}(\vec{x}, \vec{y}) = \llbracket \rho_{mn}(e, \vec{y}) \rrbracket_B^{(m)}(\vec{x})$$

for all B , e , \vec{x} , and \vec{y} . (Here equality has the usual meaning: either both sides are undefined, or both sides are defined and are the same.) Moreover, ρ_{mn} is one-to-one.

Proof: We proceed as before. Here \vec{x} is $\langle x_1, \dots, x_m \rangle$ and \vec{y} is $\langle y_1, \dots, y_n \rangle$.

$$\rho_{mn}(e, y_1, \dots, y_n) = k_{m+1}(y_1) * \dots * k_{m+n}(y_n) * k_{m+n+1}(e) * p$$

where p is the Gödel number of our program that computes $\Phi_B^{(m+n)}$. (That program did not depend on what the set B is.) (Here m , n , and p are fixed; e , \vec{x} , and \vec{y} are the variables. \dashv)

It should be noted here that ρ_{mn} is primitive recursive absolutely, and not merely primitive recursive relative to B .

Definition: A relation on \mathbb{N} is said to be *recursively enumerable relative to B* (where $B \subseteq \mathbb{N}$) if it is the domain of some partial function that is computable relative to B .

Define W_e^B to be the domain of $\llbracket e \rrbracket_B$. Then

$$W_0^B, W_1^B, W_2^B, \dots$$

is a complete list with repetitions of the sets (of natural numbers) that are recursively enumerable relative to B .

As for the unrelativized concept, there are several equivalent formulations of the definition of recursive enumerability relative to an oracle. For now, here is one of them.

Theorem: A relation R on \mathbb{N} is recursively enumerable relative to B if and only if it is Σ_1 relative to B :

$$\vec{x} \in R \iff \exists y Q(\vec{x}, y)$$

for some relation Q that is computable relative to B .

Proof: In one direction, suppose we have $\vec{x} \in R \iff \exists y Q(\vec{x}, y)$. Define the partial function:

$$f(\vec{x}) = \mu y Q(\vec{x}, y)$$

Then $R = \text{dom } f$. And f is a partial function that is computable relative to B .

In the other direction, suppose that $R = \text{dom } \llbracket e \rrbracket_B^{(n)}$. Then

$$\begin{aligned} \vec{x} \in R &\iff \exists t [\llbracket e \rrbracket_B^{(n)}(\vec{x}) \downarrow \text{ in } \leq t \text{ steps}] \\ &\iff \exists t [(\text{snap}_B^{(n)}(\vec{x}, e, t))_0 \geq \text{lh } e] \end{aligned}$$

which is Σ_1 relative to B . \dashv

Proposition: Whenever A is recursively enumerable relative to B and $B \leq_T C$, then A is recursively enumerable relative to C .

Proof: Apply the transitivity lemma to a partial function that has domain A and is computable relative to B . \dashv

For the recursively enumerable relations, our standard examples in Chapter 4 were the halting relation and the set K . The definition of K can be relativized.

Definition: For a set B of natural numbers, its *jump* is the set

$$B' = \{x \mid \llbracket x \rrbracket_B(x) \downarrow\}.$$

Theorem: (a) The set B' is recursively enumerable relative to B .

(b) B' is not computable relative to B . In fact its complement is not even recursively enumerable relative to B .

Proof: (a) B' is the domain of the function $x \mapsto \llbracket x \rrbracket_B(x)$ which by the enumeration theorem is a partial function computable relative to B .

(b) If B' were computable relative to B , then its complement would also be computable relative to B and hence recursively enumerable relative to B . So it suffices to prove the second half of part (b).

Any set that is recursively enumerable relative to B must be W_e^B for some e . But by the definition of B' we have

$$e \in \overline{B'} \iff e \notin W_e^B.$$

So $\overline{B'} \neq W_e^B$ because they differ at e . \dashv

Proposition: If $A \leq_m C$ and C is recursively enumerable relative to B , then A is also recursively enumerable relative to B .

Proof: Say f is a total computable function that many-one reduces A to C . Because C is Σ_1 relative to B , we have, for some relation Q that is computable relative to B ,

$$\begin{aligned} x \in A &\iff f(x) \in C \\ &\iff \exists y Q(f(x), y) \end{aligned}$$

and the relation $\{(x, y) \mid Q(f(x), y)\}$ is computable relative to B . \dashv

Theorem: For sets A and B of natural numbers,

$$A \text{ is recursively enumerable relative to } B \iff A \leq_1 B'.$$

Proof: The preceding proposition covers the " \Leftarrow " direction (where we take C to be B' .) The " \Rightarrow " direction asserts that B' is a 1-complete recursively enumerable set relative to B .

So assume that A is recursively enumerable relative to B , say $A = W_a^B$. Our plan is to make a one-to-one primitive recursive function g such that for each y ,

$$\begin{aligned} y \in A &\Rightarrow \llbracket g(y) \rrbracket_B \text{ is total} \Rightarrow g(y) \in B' \\ y \notin A &\Rightarrow \llbracket g(y) \rrbracket_B \text{ is empty} \Rightarrow g(y) \notin B'. \end{aligned}$$

Consider the two-place function $\langle x, y \rangle \mapsto \llbracket a \rrbracket_B(y)$ (which is a partial function $\llbracket e \rrbracket_B^{(2)}$ computable relative to B) and parameterize out y . Then taking $g(y) = \rho(e, y)$, we have the condition we sought. \dashv

In particular, taking $A = B$ in this theorem, we see that $B \leq_1 B'$. Putting this together with the fact that $B' \not\leq_T B$, we conclude that $[B] < [B']$. That is, the jump operation strictly increases the degree of a set.

Lemma: Whenever $A \leq_T B$, then $A' \leq_1 B'$.

Proof: We assume that $A \leq_T B$. We know that A' is recursively enumerable relative to A . So by a recent proposition, A' is also recursively enumerable relative to B . So by the preceding theorem, $A' \leq_1 B'$. \dashv

Applying this lemma twice, we see that whenever $A \equiv_T B$, then $A' \equiv_1 B'$, and consequently $A' \equiv_T B'$. This fact allows us to make a jump operation *on degrees*.

Definition: For a degree \mathbf{a} , define its *jump* \mathbf{a}' to be the degree $[A']$, where A is any set chosen from the degree \mathbf{a} . (The preceding lemma tells us that the degree \mathbf{a}' does not depend on which set A is chosen from \mathbf{a} .)

Because $[A] < [A']$, we can conclude that on the degrees,

$$\mathbf{a} < \mathbf{a}' < \mathbf{a}'' < \mathbf{a}''' < \dots$$

for any degree \mathbf{a} . Again, we see that there is no largest degree.

Earlier, we defined $\mathbf{0}'$ to be the degree of K . Now we are saying that $\mathbf{0}'$ is the degree of the jump of a computable set. There is no conflict here; the degrees are the same.

The converse of the preceding lemma also holds.

Proposition: Whenever $A' \leq_m B'$, then $A \leq_T B$.

Proof: On the one hand, we have $A \leq_1 A' \leq_m B'$, so A is recursively enumerable relative to B .

On the other hand, \bar{A} is recursively enumerable relative to A , and so $\bar{A} \leq_1 A' \leq_m B'$.

Hence both A and \bar{A} are recursively enumerable relative to B . By Kleene's theorem (converted to relativized form), $A \leq_T B$. \dashv

The jump operation can be thought of as providing us with a unit of measurement in the degrees. That is, a set of degree \mathbf{a}''' is "three jumps" more

complicated than a set of degree \mathbf{a} . In particular, a set of degree $\mathbf{0}'''$ is three jumps away from computability.

There is also a connection between relative computability and the arithmetical hierarchy, as in Chapter 5. The following result, here stated without proof, extends the fact that a relation is Σ_1 if and only if it is r.e.

Post's theorem: (a) A relation is Σ_2 if and only if it is r.e. relative to \emptyset' , the jump of the empty set.

(b) More generally, a relation is Σ_{k+1} if and only if it is r.e. relative to $\emptyset^{(k)}$, the k th jump of the empty set.

Exercises

6. Let $H = \{2^{u+1}3^{v+1} \mid \llbracket u \rrbracket(v) \downarrow\}$. (Thus H encodes the halting problem.) Show that the partial function

$$f(x) = \text{the least member (if any) of } W_x$$

is computable relative to H .

7. Let $H = \{2^{u+1}3^{v+1} \mid \llbracket u \rrbracket(v) \downarrow\}$. (Thus H encodes the halting problem.) Show that the partial function

$$g(x) = \text{the least member (if any) of } \overline{W_x}$$

is computable relative to H .

8. Let $\text{Fin} = \{x \mid W_x \text{ is finite}\}$. Let

$$B = \{2^{x+1}3^{y+1} \mid W_x \text{ contains a number } \geq y\}.$$

Show that Fin is recursively enumerable relative to B .

9. Let $R = \{x \mid W_x \text{ is a computable set}\}$. Let

$$C = \{2^{u+1}3^{v+1} \mid W_u \text{ and } W_v \text{ are complementary}\}.$$

Show that R is recursively enumerable relative to C .

10. Show that $\overline{\text{Tot}}$ is recursively enumerable relative to K .

11. Give an example of sets A , B , and C for which A is recursively enumerable relative to B , B is recursively enumerable relative to C , but A is *not* recursively enumerable relative to C . (The point of this exercise is to show that relative recursive enumerability is *not* transitive.)

12. Call a set S of degrees *large* if it includes, for some degree \mathbf{a} , the entire "cone" $\{\mathbf{c} \mid \mathbf{a} \leq \mathbf{c}\}$. Show that the intersection of any two large sets of degrees is again large.

13. For each of the following sets of degrees, determine whether or not it is large, in the sense of the preceding exercise. Also determine whether or not its complement is large.

- (a) The set of recursively enumerable degrees.
- (b) The set of degrees \mathbf{a} such that every set in \mathbf{a} is infinite.

14. Let $H = \{2^{u+1}3^{v+1} \mid \llbracket u \rrbracket(v) \downarrow\}$. (Thus H encodes the halting problem.) Show that there is a total function f computable relative to H that dominates every total computable function g (in the sense that $f(x) > g(x)$ for all but finitely many values of x).