

Notes on Computability

Anagha

Introduction

These notes introduce the basics of computability theory: partial and total computable functions, diagonalization, recursion theorems, computably enumerable sets, Turing reducibility, and the arithmetical hierarchy. They closely follow Rebecca Weber's *Computability Theory* [1]. The presentation is concise; some standard theorems are stated without proof, while examples and selected proofs illustrate the main techniques. Exercise numbers retained below refer to Weber's book.

Throughout, $\mathbb{N} = \{0, 1, 2, \dots\}$. Unless another domain is specified, sets are subsets of $\mathbb{N}$ and functions take values in $\mathbb{N}$. We use *computable* and *recursive* interchangeably, and *computably enumerable* (c.e.) and *recursively enumerable* (r.e.) interchangeably. Complements are taken in $\mathbb{N}$. Acknowledgements appear at the end.

1 Defining computability

1.1 Partial functions and convergence

A partial function $f : X \rightarrow Y$ is a function whose domain of definition $\text{dom}(f)$ is a subset of X . For each $x \in X$, either there is a unique $y \in Y$ with $f(x) = y$, or $f(x)$ is undefined. The function is *total* if $\text{dom}(f) = X$.

We write $f(x) \downarrow$ when $f(x)$ is defined and $f(x) \uparrow$ when it is undefined. For a partial computable function, these statements mean that its computation halts or fails to halt, respectively. Here *convergence* refers to halting, rather than to the existence of a numerical limit. An arbitrary natural-number-valued sequence need not converge. If such a sequence has a finite real limit, it is eventually constant; the limit approximations used later have this property for each fixed input.

Two partial functions are equal when they have the same domain and agree there:

$$f = g \iff \text{dom}(f) = \text{dom}(g) \text{ and } (\forall x \in \text{dom}(f)) f(x) = g(x).$$

An equation between partial computations below means that both sides are undefined, or both are defined with the same value.

For $A \subseteq \mathbb{N}$, its *characteristic function* is the total function

$$\chi_A(n) = \begin{cases} 1 & n \in A, \\ 0 & n \notin A. \end{cases}$$

1.2 Turing machines

A Turing machine (TM) has a finite set of states, a finite tape alphabet containing a blank symbol, a designated initial state, a halting convention, and a finite deterministic transition table. Its tape is unbounded, although every halting computation visits only finitely many cells.

One convenient convention describes an *individual instruction* by a quadruple $\langle q, a, \alpha, q' \rangle$. In state q , reading symbol a , the machine performs action α and enters state q' . The action is either to write a tape symbol without moving, or to move one cell left or right without writing. Writing and moving may therefore require two instructions. The states q and q' need not be initial or final states. A whole machine consists of a finite compatible collection of instructions, together with the other data above. We write h for a halting state with no outgoing instructions, and B for the blank symbol.

Exercises

1. Flip the bits of a finite binary string. Place the head at the first bit. Use states q, r, h and the rules

$$\begin{aligned} \langle q, 0, 1, r \rangle, \quad \langle q, 1, 0, r \rangle, \quad \langle q, B, B, h \rangle, \\ \langle r, 0, R, q \rangle, \quad \langle r, 1, R, q \rangle. \end{aligned}$$

Thus B011010B becomes B100101B. The first two rules flip a bit; the last two move to the next cell.

2. Add one in tally notation. Represent n by 1^n and start at its leftmost cell (a blank when $n = 0$). The rules

$$\langle q, 1, R, q \rangle, \quad \langle q, B, 1, h \rangle$$

scan to the first blank, write one more tally mark, and halt.

3. Binary addition of one. Write the binary digits with the *least significant bit first*, and place the head on that bit. The rules

$$\begin{aligned} \langle q, 0, 1, h \rangle, \quad \langle q, 1, 0, r \rangle, \\ \langle r, 0, R, q \rangle, \quad \langle q, B, 1, h \rangle \end{aligned}$$

propagate the carry to the right. For example, the string 110, representing 3, becomes 001, representing 4. With the usual most significant bit first convention, the head would instead start at the rightmost digit and propagate the carry leftward.

1.3 Primitive recursive functions

The primitive recursive functions are the smallest class of total functions of finite arity containing the successor function $S(x) = x + 1$, all constant functions $C_m^n(x_1, \dots, x_n) = m$, and all projections $P_i^n(x_1, \dots, x_n) = x_i$ for $n \geq 1$ and $1 \leq i \leq n$, and closed under:

1. **Composition:** from $g_1, \dots, g_k$ and h , form $f(\vec{x}) = h(g_1(\vec{x}), \dots, g_k(\vec{x}))$.
2. **Primitive recursion:** from g, h , form the unique f satisfying

$$f(\vec{x}, 0) = g(\vec{x}), \quad f(\vec{x}, y + 1) = h(\vec{x}, y, f(\vec{x}, y)).$$

It is the functions themselves, rather than their individual values, that belong to this class.

Examples

1. Addition and multiplication. Addition is primitive recursive by

$$a(x, 0) = x, \quad a(x, y + 1) = S(a(x, y)).$$

Consequently multiplication is primitive recursive by

$$m(x, 0) = 0, \quad m(x, y + 1) = a(m(x, y), x).$$

2. Exponentiation.

$$p(x, 0) = 1, \quad p(x, y + 1) = m(x, p(x, y)).$$

This convention gives $0^0 = 1$.

3. Factorials.

$$F(0) = 1, \quad F(n + 1) = (n + 1)F(n).$$

4. Routes on a rectangular grid. Suppose $n \geq 1$ east-west streets cross $m \geq 1$ north-south streets. To go from the northwest intersection to the southeast intersection requires $n - 1$ southward moves and $m - 1$ eastward moves. Therefore

$$r(n, m) = \binom{n + m - 2}{n - 1}.$$

The boundary conditions and recurrence are

$$r(1, m) = r(n, 1) = 1, \quad r(n, m) = r(n - 1, m) + r(n, m - 1) \quad (n, m \geq 2).$$

The recurrence follows by considering the final move. To see explicitly that r is primitive recursive, its binomial formula is the quotient of $(n + m - 2)!$ by $(n - 1)!(m - 1)!$. This integer quotient can be found by a search bounded by $(n + m - 2)!$; bounded search on a primitive recursive predicate is primitive recursive. Set $r(n, m) = 0$ if either argument is zero to obtain a total function on $\mathbb{N}^2$. If n, m instead count *moves*, the corresponding formula is $\binom{n+m}{n}$.

5. Minimum and maximum. Truncated subtraction $x \dot{-} y = \max\{x - y, 0\}$ is primitive recursive, using the primitive recursive predecessor function. Hence so are

$$\min(x, y) = x \dot{-} (x \dot{-} y), \quad \max(x, y) = x + (y \dot{-} x).$$

1.4 A total computable function that is not primitive recursive

The Ackermann function is defined by

$$A(m, n) = \begin{cases} n + 1 & m = 0, \\ A(m - 1, 1) & m > 0, n = 0, \\ A(m - 1, A(m, n - 1)) & m > 0, n > 0. \end{cases}$$

Nested induction, first on m and then on n , shows that this recursive algorithm terminates on every input. It is therefore total computable. The following standard domination theorem is stated without proof:

Proposition. *For every unary primitive recursive function θ , there is an N such that $A(m, m) > \theta(m)$ for all $m > N$.*

In particular A cannot be primitive recursive, since otherwise $m \mapsto A(m, m)$ would be a primitive recursive function contradicting the proposition applied to itself.

1.5 Partial recursive functions and unbounded search

The partial recursive functions are the smallest class containing the initial functions above and closed under composition, primitive recursion, and *unbounded minimization*. In this definition the closure operations may be applied to partial functions, with the required intermediate computations defined.

For a partial function $\theta(\vec{x}, y)$, define

$$\mu y [\theta(\vec{x}, y) = 0]$$

to be the least y for which $\theta(\vec{x}, y) = 0$ and, for every $z < y$, $\theta(\vec{x}, z)$ is defined and nonzero. If no such y exists, the result is undefined. Operationally, compute $\theta(\vec{x}, 0), \theta(\vec{x}, 1), \dots$ in order, stopping at the first zero. A divergent earlier computation prevents the search from reaching later candidates. For a total primitive recursive θ , only the possibility of finding no zero causes divergence.

For example $\mu y [y > 5] = 6$, with the predicate convention that the search seeks the least satisfying value. If

$$\theta(x, y) = \begin{cases} 0 & x \text{ is even,} \\ 1 & x \text{ is odd,} \end{cases}$$

then $\mu y [\theta(x, y) = 0]$ is 0 for even x and is undefined for odd x . Divergence is not a value ∞ in $\mathbb{N}$.

2 Coding, countability, and universal computation

2.1 Coding and effective enumeration

A coding of a collection of finite objects assigns natural-number codes so that the relevant syntactic operations can be performed effectively. Mere existence of a set-theoretic bijection with $\mathbb{N}$ does not provide an effective coding, and a finite set cannot be in bijection with $\mathbb{N}$. A machine can decode its input, compute on the represented object, and encode the output, or work directly on the codes.

We avoid using *effectively countable* for ordinary countability. For a subset of an already effectively coded domain, *effectively enumerable* means that a procedure lists its members; equivalently, its set of codes is c.e. Every c.e. set is countable, while an arbitrary subset of $\mathbb{N}$ need not be c.e. Every nonempty c.e. set has a total computable enumeration, possibly with repetitions. For an infinite c.e. set, repetitions can be removed effectively to give a one-to-one enumeration; its order need not be increasing.

Example (Integers). An effective bijection from $\mathbb{Z}$ to $\mathbb{N}$ is

$$c(k) = \begin{cases} 2k & k \geq 0, \\ -2k - 1 & k < 0. \end{cases}$$

Even codes decode to $k = c/2$, and odd codes decode to $k = -(c+1)/2$.

2.2 Pairing and finite tuples

The Cantor pairing function

$$\langle x, y \rangle = \pi(x, y) = \frac{(x+y)(x+y+1)}{2} + x$$

is a computable bijection from $\mathbb{N}^2$ to $\mathbb{N}$ with computable inverse. The diagonals $x + y = d$ occupy consecutive intervals of codes. Given a code, a bounded search determines its diagonal and then its two coordinates. Iterating pairing, for example $\langle x, y, z \rangle = \langle \langle x, y \rangle, z \rangle$, gives effective codings of $\mathbb{N}^k$ for each fixed $k \geq 1$. This is the same diagonal idea used in proving that $\mathbb{Q}$ is countable.

Unions and intersections

If $A = \{a_0, a_1, \dots\}$ and $B = \{b_0, b_1, \dots\}$ have computable total enumerations, then

$$a_0, b_0, a_1, b_1, \dots$$

enumerates $A \cup B$. This works whether or not the sets are disjoint; repeated outputs may be suppressed. Listing all of A *before* starting B would never reach B if A is infinite.

The intersection $A \cap B$ is countable because it is a subset of a countable set. In fact, the intersection of two c.e. sets is c.e.: dovetail their enumerations and output an element when it has appeared in both. This does not make arbitrary subsets or differences of c.e. sets c.e.

2.3 Indices, padding, and a universal machine

Fix a standard effective enumeration of machine programs. Write φ_e for the partial function computed by program e , and set

$$W_e = \text{dom}(\varphi_e).$$

An *index* is a code of one program; the whole sequence $(\varphi_e)_{e \in \mathbb{N}}$ is an enumeration of the partial computable functions. A function can have many different indices.

Lemma (Padding). *For every index e there are arbitrarily large indices e' with $\varphi_{e'} = \varphi_e$. Such indices can be produced effectively by adding unused instructions to a program, under the standard syntactic coding.*

A universal Turing machine computes the partial function

$$U(\langle e, x \rangle) = \varphi_e(x).$$

It simulates program e on input x , returning its output if the simulation halts and diverging if the simulation diverges.

Theorem. *There is no total computable function h such that every $\varphi_{h(n)}$ is total and every total computable function appears among the $\varphi_{h(n)}$. Consequently $\text{Tot} = \{e : \varphi_e \text{ is total}\}$ is not c.e.*

Proof. If h existed, $g(n) = \varphi_{h(n)}(n) + 1$ would be total computable. If $g = \varphi_{h(k)}$, then $g(k) = g(k) + 1$, a contradiction. If Tot were c.e., its nonemptiness would give a total computable enumeration h of its elements, contradicting the first statement. $\square$

Thus countability of the total computable functions does not supply a computable listing of them as total functions.

2.4 The Church–Turing thesis and equivalent models

Theorem. *A partial function is Turing computable if and only if it is partial recursive.*

This is a mathematical equivalence between two formal definitions. The *Church–Turing thesis* is the further claim that these formal definitions capture the informal notion of effective calculation. The thesis itself is not the formal equivalence theorem.

For example, a TM with a two-symbol tape alphabet can simulate one with any fixed finite alphabet. Choose a fixed-length binary code for its symbols, taking the blank symbol to an all-zero block. The simulator tracks block alignment in its finite control, reads or changes one encoded block, and moves by one block for each simulated head move. Finite input is supplied in the corresponding encoding.

Other equivalent models use a semi-infinite tape, several work tapes, several heads, a two-dimensional grid, or moves of any fixed finite length. Each variant has an effective simulation by an ordinary TM; these equivalences concern which functions can be computed, rather than the time taken.

In a nondeterministic TM, a configuration can have several possible successors. With the usual acceptance convention, an input is accepted if some finite branch accepts. A deterministic machine can search the finitely branching computation tree by breadth-first search. It therefore finds every accepting branch that exists, so nondeterministic and deterministic machines recognize the same languages.

3 Working with computable functions

3.1 The halting problem and diagonalization

Define the diagonal halting set and its characteristic function by

$$K = \{e : \varphi_e(e) \downarrow\}, \quad f(e) = \chi_K(e) = \begin{cases} 1 & \varphi_e(e) \downarrow, \\ 0 & \varphi_e(e) \uparrow. \end{cases}$$

If f were computable, so would be the total function

$$g(e) = \begin{cases} \varphi_e(e) + 1 & f(e) = 1, \\ 1 & f(e) = 0. \end{cases}$$

In the first case, the positive answer from f guarantees that the simulation eventually halts. Let d be an index for g . Since g is total, $\varphi_d(d) \downarrow$, so $g(d) = \varphi_d(d) + 1 = g(d) + 1$, a contradiction. Thus K is not computable. It is nevertheless c.e., by dovetailing the computations $\varphi_e(e)$.

The underlying diagonal argument is worth isolating. Given a sequence of partial functions (ψ_n) , define the partial function $d(n) = \psi_n(n) + 1$. If d is total, then d differs from each ψ_n at input n , so it cannot occur in that sequence. For an effective list of *total* computable functions, this would make d a missing total computable function. For a list of all partial computable functions, the diagonal construction need not be total.

3.2 Parametrization: the s-m-n theorem

Theorem. For all $m, n \geq 1$, there is a total computable function s_m^n of $m + 1$ arguments such that

$$\varphi_{s_m^n(e, \vec{x})}(\vec{y}) = \varphi_e(\vec{x}, \vec{y}), \quad \vec{x} \in \mathbb{N}^m, \quad \vec{y} \in \mathbb{N}^n.$$

For the standard syntactic numbering, s_m^n can be chosen primitive recursive and injective in the full tuple $(e, \vec{x})$.

The subscript here counts the parameters fixed in advance, and the superscript counts the remaining arguments. The important strengthening is *injectivity*; surjectivity onto all program indices is not asserted.

The theorem compiles the values $\vec{x}$ into a program while leaving $\vec{y}$ as its input. In particular,

$$\varphi_{s_1^1(e,x)}(y) = \varphi_e(x, y).$$

If $F(x, y)$ is any partial computable function, fixing an index for F gives a total computable s with $\varphi_{s(x)}(y) = F(x, y)$. Together with universality, parametrization turns effective operations on functions into effective operations on their indices.

3.3 The fixed point theorem

Theorem (Kleene's recursion theorem). *If $f : \mathbb{N} \rightarrow \mathbb{N}$ is total computable, there is an index e such that $\varphi_e = \varphi_{f(e)}$. An index e can be found uniformly from an index for f , provided the latter computes a total function.*

This is a fixed point of the *computed function*, not necessarily an equality $e = f(e)$ of numbers. The theorem is stated without proof here.

Corollary 4.4.4. If $F(x, y)$ is partial computable, there is an e with $\varphi_e(y) = F(e, y)$ for every y , in the sense of equality of partial computations. Indeed, choose total computable f with $\varphi_{f(x)}(y) = F(x, y)$ and apply the theorem to f .

3.4 Rice's theorem and index sets

A set $I \subseteq \mathbb{N}$ is an *index set* if

$$\varphi_e = \varphi_d \implies (e \in I \iff d \in I).$$

It describes a property of the computed partial function rather than of a particular program text. It is *nontrivial* if $I \neq \emptyset$ and $I \neq \mathbb{N}$.

Theorem (Rice). *Every nontrivial index set is noncomputable.*

Proof. First suppose an index for the everywhere undefined function is not in I . Choose $a \in I$. Uniformly in x , construct a program which, on input y , waits for $\varphi_x(x)$ to halt and then runs $\varphi_a(y)$. Its index $h(x)$ is computable by parametrization, and $x \in K$ if and only if $h(x) \in I$. Thus a decision procedure for I would decide K . If the everywhere undefined function belongs to I , apply the same argument to its nontrivial complement. $\square$

In particular, nontrivial semantic properties of the language recognized by a machine are undecidable. Properties of program syntax need not be undecidable.

The general halting set is

$$K_0 = \{\langle e, x \rangle : \varphi_e(x) \downarrow\}.$$

We have $e \in K \iff \langle e, e \rangle \in K_0$; conversely, compile (e, x) into a program ignoring its own input and simulating $\varphi_e(x)$. This reduces K_0 to K .

Examples of noncomputable index sets, with $W_e = \text{dom}(\varphi_e)$, are

$$\begin{aligned} \text{Fin} &= \{e : W_e \text{ is finite}\}, & \text{Inf} &= \{e : W_e \text{ is infinite}\}, \\ \text{Tot} &= \{e : W_e = \mathbb{N}\}, & \text{Rec} &= \{e : W_e \text{ is computable}\}. \end{aligned}$$

Each property is invariant under equality of partial functions and has both positive and negative examples. For Rec, the sets $\mathbb{N}$ and K supply such examples. For decision problems, *decidable*, *solvable*, and *computable* refer to the computability of the associated characteristic function.

3.5 Production systems

A semi-Thue system over a finite alphabet consists, in the usual finite presentation, of finitely many word-rewriting rules $u \rightarrow v$. A rule permits $PuQ \rightarrow PvQ$ in any context P, Q . We may take u, v nonempty for the examples here. With a designated axiom word a , a word is a *theorem* if it is obtainable from a by finitely many rewrites.

Finite derivations can be enumerated, so the set of theorems is c.e. A Turing machine can be encoded in a finite semi-Thue system. Consequently the general derivability problem is undecidable; there are even fixed finite systems and axioms with undecidable theorem sets. This does not mean every particular semi-Thue system has an undecidable theorem set.

One can also consider an effectively specified infinite rule set. For example, the rules $01^n0 \rightarrow 01^{n+1}0$ for $n \in \mathbb{N}$, with axiom 00 , generate precisely $\{01^n0 : n \in \mathbb{N}\}$. This is an infinite collection of productions with a decidable theorem set. For an arbitrary infinite collection without an effective presentation, the assertion that all derivations can be effectively enumerated would require an additional assumption.

3.6 Selected exercises

4.1.2. A busy-beaver time bound. Fix a finite tape alphabet, the TM convention, and a blank initial tape. Let $T(n)$ be the largest halting time among machines with states drawn from $\{q_0, \dots, q_n\}$. Use $T(n) = 0$ if no machine in the class halts. There are only finitely many such transition tables, so the maximum exists. Specifying the blank input is essential: arbitrary unbounded inputs would not give a finite bound for a fixed state count.

There is no total computable function B with $B(n) \geq T(n)$ for every n . To prove this, given a machine M and a finite input w , effectively construct a blank-input machine M_w that first writes w and then simulates M on w . Compute n large enough to contain all its states, and simulate M_w for $B(n)$ steps. If it has not halted by then, it never halts, by the definition of $T(n)$. This would decide the general halting problem. Taking $B = T$ also proves that T is not computable. Uncomputability alone would not have excluded a computable upper bound; the bounded simulation argument proves the stronger claim.

4.3.2. Multiplying the output of a program by two. Define $F(x, y)$ by simulating $\varphi_x(y)$ and returning twice its output when it halts. If that simulation diverges, so does $F(x, y)$. Universality makes F partial computable, and the s - m - n theorem gives a total computable f with

$$\varphi_{f(x)}(y) = 2\varphi_x(y)$$

for every x, y , including agreement on divergence.

4 Computing and enumerating sets

4.1 Dovetailing

Suppose we want to run a computable family of potentially divergent computations. Running the first to completion before starting the second can prevent all later work. Instead, at stage s , simulate each of the first $s + 1$ computations for s steps, recording outputs already seen. Alternatively, preserve their configurations and give each active computation another step at each later stage. Every individual stage takes finite time, and every computation that halts is eventually observed. It is *not* necessary that every computation halt.

This resembles breadth-first search in a computably presented finitely branching tree: every finite level can be searched in finite time. Depth-first search can remain on an infinite branch and miss other nodes.

4.2 Computable and computably enumerable sets

A set A is computable if χ_A is computable. The complement of a computable set is computable, since $\chi_{\bar{A}} = 1 - \chi_A$. Every finite set is computable, by comparing the input against its finitely many members. These statements are stronger than mere countability.

Theorem. For $A \subseteq \mathbb{N}$, the following are equivalent:

1. A is c.e.: a procedure enumerates its elements, with repetitions allowed.
2. A is the domain of a partial computable function.
3. A is the range of a partial computable function.
4. $A = \emptyset$, or A is the range of a total computable function.
5. There is a total computable $f : \mathbb{N}^2 \rightarrow \{0, 1\}$ such that $f(x, 0) = 0$, $f(x, s) \leq f(x, s+1)$, and $\lim_s f(x, s) = \chi_A(x)$ for every x . Thus each input changes its value at most once.
6. There is a uniformly computable sequence of finite sets (A_s) , with computable finite codes, such that $A_s \subseteq A_{s+1}$ and $A = \bigcup_s A_s$.

Proof sketch. From an enumerator, a program halts on x when x is listed. Conversely, dovetail a partial program on all inputs and enumerate the inputs on which it halts, or their outputs, as appropriate. A partial range is therefore c.e.; an enumerator also yields a partial function whose successive outputs are its range. Normalize the enumerator to output at most one number per step. If the set is nonempty, wait for the first output at a step t_0 ; on input s , run it for $\max\{s, t_0\}$ steps and return its most recent output. This is a total computable enumeration of the entire set, with repetition allowed. The empty set has empty partial domain and range but cannot be a total range.

For the last two clauses, run an enumeration for s steps and collect the finitely many elements seen into A_s , starting with $A_0 = \emptyset$. Then take $f(x, s) = \chi_{A_s}(x)$. Conversely, enumerate the computably given finite sets A_s , or enumerate x when its computable approximation changes to 1. $\square$

Uniform computability in clause 6 is indispensable: every subset of $\mathbb{N}$ is an increasing union of finite sets if no effectiveness condition is imposed.

Proposition. An infinite set A is computable if and only if it is the range of a strictly increasing total computable function.

Proof. If A is computable and infinite, set $g(0) = \min A$ and $g(n+1) = \min\{x > g(n) : x \in A\}$. Membership tests are computable, and infinitude ensures that every search terminates. Conversely, given such g , on input x compute $g(0), g(1), \dots$ until the first value at least x appears. Its existence follows because a strictly increasing sequence of natural numbers is unbounded. Accept exactly when that value equals x . $\square$

The characteristic function of A is not in general increasing, and an enumeration of an infinite set does not finish after a finite time.

Lemma. *Every infinite c.e. set has an infinite computable subset.*

Proof. From an enumeration of A , choose a first element c_0 , then wait for an element $c_1 > c_0$, then for $c_2 > c_1$, and so on. Infinitude makes this a total computable strictly increasing sequence. Its range is computable by the preceding proposition and is a subset of A . $\square$

Exercise 5.2.9. A set A is computable if and only if both A and $\bar{A}$ are c.e. One direction follows from closure of computability under complementation. For the other, on input x run recognizers for A and $\bar{A}$ by dovetailing. Exactly one eventually accepts. Accept if the first recognizer accepts and reject if the second accepts. This always terminates.

4.3 Enumeration and incompleteness

A first-order language specifies constant, function, and relation symbols with their arities, in addition to logical symbols and variables. Formulas are built from these symbols; sentences are formulas without free variables. A structure has a nonempty underlying domain and interprets the nonlogical symbols as elements, functions, and relations on that domain. It is a model of a theory T if every sentence of T is true in it.

The soundness and completeness theorems for first-order logic state

$$T \vdash \phi \iff T \models \phi.$$

In particular, a sentence is logically valid if and only if it is provable from the logical axioms. For an effectively presented language, valid sentences are c.e. by proof enumeration. In sufficiently expressive languages, first-order validity is not decidable (Church's theorem). This undecidability assertion does not apply to every restricted language or fragment, such as propositional logic.

A sentence ϕ is independent of T if neither $T \vdash \phi$ nor $T \vdash \neg\phi$. By completeness, this is equivalent to the existence of models $M, N \models T$ with $M \models \phi$ and $N \models \neg\phi$.

Robinson arithmetic

Robinson arithmetic, usually denoted Q , is a finitely axiomatized theory in the language $\{0, S, +, \cdot, =\}$. Its axioms are the universal closures of

$$\begin{aligned} Sx \neq 0, \quad Sx = Sy &\longrightarrow x = y, \quad x \neq 0 \longrightarrow (\exists y) x = Sy, \\ x + 0 = x, \quad x + Sy &= S(x + y), \\ x \cdot 0 = 0, \quad x \cdot Sy &= x \cdot y + x. \end{aligned}$$

One may add $<$ as a defined relation, for example $x < y \iff (\exists z) x + S(z) = y$. The standard natural numbers form a model of Q , so it is consistent.

The set P of sentences provable in Q is c.e. So is $R = \{\phi : Q \vdash \neg\phi\}$: enumerate proofs and sentences and match proofs whose conclusions are their negations. Consistency gives $P \cap R = \emptyset$. The incompleteness theorem for consistent effectively axiomatized extensions of Q implies that Q has independent sentences; this theorem is stated here without proof. Completeness of first-order logic and incompleteness of Q refer to different assertions and are compatible.

4.4 Simple and immune sets

A noncomputable set is a set whose characteristic function differs from every total computable function. Such a set need not be c.e. or co-c.e. We now focus on a particular class of noncomputable c.e. sets.

Definition. A set is *immune* if it is infinite and contains no infinite c.e. subset. A set A is *simple* if A is c.e. and $\overline{A}$ is immune.

Thus a simple set is coinfinite, and every infinite c.e. set meets it. Its complement is immune, not simple. Every simple set is infinite and noncomputable: if A were computable (including the case that it is finite), its infinite complement would itself be an infinite c.e. subset of $\overline{A}$, contradicting immunity.

Proposition. A coinfinite c.e. set A is simple if and only if it is contained in no coinfinite computable set.

Proof. If $A \subseteq C$ and C is computable and coinfinite, then $\overline{C}$ is an infinite computable subset of $\overline{A}$. Conversely, if A is not simple, an infinite c.e. subset of $\overline{A}$ has an infinite computable subset D . Then $C = \overline{D}$ is a coinfinite computable superset of A . $\square$

If A and B are simple, then $A \cap B$ is simple. To see this, an infinite c.e. set W meets A in infinitely many elements: otherwise deleting the finite set $W \cap A$ leaves an infinite c.e. subset of $\overline{A}$. Thus $W \cap A$ is an infinite c.e. set and meets B . Also, $\overline{A \cap B}$ is infinite because it contains $\overline{A}$. Likewise $A \cup B$ is either simple or cofinite: its complement is a subset of the immune set $\overline{A}$, and hence is immune whenever it is infinite.

5 Turing reductions and Post's problem

5.1 Oracle computations and their use

An oracle TM with oracle A can query whether a number n belongs to A and receive the correct answer in one oracle step. The same program index specifies a partial function φ_e^A for every oracle A . A halting computation makes only finitely many queries, although the number and positions queried can depend on its input and the oracle's answers. A divergent computation can make infinitely many queries. Finitely many oracle queries in a run do not make the oracle itself computable.

Write $\varphi_{e,s}^A(x) \downarrow = y$ if the computation halts within s steps with output y . For a finite binary string σ , a query $n < |\sigma|$ is answered by $\sigma(n)$; a query $n \geq |\sigma|$ makes that finite-oracle computation undefined.

Exercise. The set

$$\{ \langle \sigma, e, x, s, y \rangle : \varphi_{e,s}^\sigma(x) \downarrow = y \}$$

is computable. Decode the tuple and simulate at most s steps, answering from σ . Reject if the simulation asks an unavailable query, fails to halt within the bound, or halts with a different output. Accept exactly if it halts within the bound with output y . No unbounded waiting or dovetailing is needed.

For a convergent computation, define its *use* by

$$u(A; e, x) = \max(\{n + 1 : n \text{ is queried during the computation}\} \cup \{0\}).$$

This includes queries answered *no*, as well as those answered *yes*, and gives use 0 when there are no queries. For stage-bounded computations, write $u(A; e, x, s)$ when convergence occurs within s steps, and optionally assign 0 otherwise as a bookkeeping convention. A divergent unbounded computation has no convergent-computation use under this convention.

Identify A with its characteristic sequence, and let $A \upharpoonright n$ be the finite string of its first n bits. Then the *use principle* says

$$\varphi_e^A(x) \downarrow = y \implies (\exists s)(\exists n) \varphi_{e,s}^{A \upharpoonright n}(x) \downarrow = y.$$

Moreover, if $\varphi_{e,s}^\sigma(x) \downarrow = y$, then for every $t \geq s$ and every finite or infinite oracle X extending σ ,

$$\varphi_{e,t}^X(x) \downarrow = y.$$

The computation sees the same finite list of answers in each extension.

5.2 Turing reducibility and completeness

Define

$$A \leq_T B \iff (\exists e) \varphi_e^B = \chi_A.$$

Thus A is computable with oracle B . Define $A \equiv_T B$ if both $A \leq_T B$ and $B \leq_T A$. They are *Turing incomparable*, written $A \perp_T B$, if neither reduction holds.

Turing reducibility is reflexive and transitive. Transitivity is uniform: there is a total computable k such that if

$$\chi_C = \varphi_e^B \quad \text{and} \quad \chi_B = \varphi_i^A, \quad \text{then} \quad \chi_C = \varphi_{k(e,i)}^A.$$

Simulate the outer program and answer its B -queries using the inner program with oracle A . The inner computations terminate because they compute χ_B . Consequently $\equiv_T$ is an equivalence relation. Every computable set is reducible to every oracle by ignoring the oracle. If A is computable and $B \leq_T A$, then B is computable, since oracle answers can be computed directly.

A c.e. set is *Turing complete for the c.e. sets* if every c.e. set is Turing reducible to it. The halting set K is complete: for $A = W_e$, an oracle for $K_0 \equiv_T K$ decides whether $\varphi_e(x)$ halts.

Another complete c.e. set, sometimes called the weak jump here, is

$$H = \{e : W_e \neq \emptyset\}.$$

Its enumerability follows by searching for any convergent computation in each domain. To reduce K to H , compile e into a program that, on every input, waits for $\varphi_e(e)$ to halt and then halts. Its domain is $\mathbb{N}$ or $\emptyset$ according as $e \in K$ or $e \notin K$. Conversely, to decide $e \in H$ with K , ask whether the search for a convergent computation of φ_e eventually halts. Hence $H \equiv_T K$.

5.3 Constructing a simple set

Fix uniformly computable finite approximations $W_{e,s}$ with $W_{e,s} \subseteq W_{e,s+1}$ and $W_e = \bigcup_s W_{e,s}$. We construct a c.e. set A meeting the requirements

$$R_e : |W_e| = \infty \implies (\exists x > 2e) x \in A \cap W_e.$$

Start with $A_0 = \emptyset$. At stage s , for each $e \leq s$ that has not yet acted, if $W_{e,s}$ contains an $x > 2e$, put one such x into A and declare R_e finished. Each R_e enumerates at most one number and meets its requirement whenever W_e is infinite.

Only requirements R_e with $e < n$ can enumerate a number at most $2n$. They contribute at most n numbers in total, leaving at least $n+1$ numbers outside A in $\{0, \dots, 2n\}$. Thus $\bar{A}$ is infinite. An infinite c.e. subset of $\bar{A}$ would be some W_e and would have to meet A , a contradiction. Therefore A is simple. This elementary construction does not yet require injury between competing strategies.

5.4 Finite injury and Post's problem

When requirements compete, assign a priority order. A higher-priority requirement R_i may injure a lower-priority requirement R_j only when $i < j$. *Finite injury* is a conclusion that must be verified: each requirement is injured only finitely often, so its final strategy eventually settles.

Post's problem asks whether there is a c.e. set A with

$$\emptyset <_T A <_T K.$$

We give a finite injury construction of a simple A and a c.e. set B such that $B \not\leq_T A$. Simplicity ensures that A is noncomputable; the second property ensures that it is incomplete, since a complete c.e. set would compute the c.e. set B .

Use the requirements R_e above together with

$$Q_e : \quad \varphi_e^A \neq \chi_B, \quad R_0 > Q_0 > R_1 > Q_1 > \dots$$

An R_e strategy may enumerate *at most one* number over the entire construction. It waits for $x \in W_{e,s}$ with $x > 2e$ and x at least every restraint currently imposed by a higher-priority Q strategy. It then enumerates x into A and is permanently finished. The bound on its total number of enumerations is retained even when other strategies are initialized.

Each Q_e receives a fresh witness b_e that has never been used by any strategy, and hence is outside the current B . It waits for $\varphi_{e,s}^{A_s}(b_e) \downarrow = v$. On seeing convergence, it preserves a disagreement by putting b_e into B if $v = 0$, and leaving it out if $v > 0$. It imposes restraint equal to the use of that computation: lower-priority R strategies may subsequently enumerate into A only at or above this restraint. The Q_e strategy then stops acting until injured. A later change in A by a higher-priority R strategy initializes lower-priority Q strategies, removing their old restraints and giving them fresh witnesses. Numbers already in A or B are never removed; old witnesses are never reused.

Effective stage procedure. At stage s , initialize any strategies of index at most s that lack a witness. Among the strategies of index at most s whose conditions for action are visible in $W_{e,s}$ and the s -step simulations, let the highest-priority eligible strategy act. If an R strategy acts, initialize all lower-priority Q strategies. All tests are bounded, so this enumerates c.e. sets A, B .

Verification. Induct on priority. Each R_e acts at most once. Each Q_e is initialized only by the finitely many higher-priority R strategies, and acts at most once between initializations. Hence all higher-priority restraints eventually stabilize and every strategy suffers only finitely many injuries. After stabilization, an infinite W_e eventually supplies a number greater than $2e$ and all relevant restraints, so R_e acts if it has not already acted. Highest-priority scheduling cannot postpone it forever because only finitely many higher-priority actions remain. As in the preceding construction, the one-enumeration bound for R_e keeps $\bar{A}$ infinite, and all R_e together make A simple.

After the last initialization of Q_e , if it acts, its restraint is respected forever and the use principle preserves its disagreement with χ_B . If it never acts and $\varphi_e^A(b_e)$ were convergent, the finite portion of A_s used by that final computation would eventually stabilize. A sufficiently late stage would then see the convergence and permit Q_e to act, a contradiction. Thus in this case φ_e^A is not total. Every Q_e is satisfied, and $B \not\leq_T A$, as required. This proves the affirmative answer to Post's problem. The stronger Friedberg–Muchnik theorem gives two Turing incomparable c.e. sets.

6 Hierarchies of sets

6.1 Turing degrees

The *Turing degree* of a set A is its equivalence class

$$\deg_T(A) = \{B : B \equiv_T A\}.$$

The quotient $\mathcal{P}(\mathbb{N})/\equiv_T$ is the *set of all Turing degrees*, not one degree. The induced order is $\deg_T(A) \leq \deg_T(B)$ if $A \leq_T B$; it is independent of the chosen representatives by transitivity.

1. Least degree. The least degree is $\mathbf{0} = \deg_T(\emptyset)$, consisting exactly of the computable sets. Every computable set reduces to every oracle.

2. Joins. Define

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

Both A and B reduce to $A \oplus B$, by asking even or odd queries. If $A \leq_T C$ and $B \leq_T C$, a C -oracle decides membership in their join by checking parity and using the appropriate reduction. Thus $\deg_T(A \oplus B)$ is the least upper bound of their degrees.

3. Complements. For every A , $A \equiv_T \bar{A}$, by reversing the answer to each membership query. Hence $\deg_T(A) = \deg_T(\bar{A})$.

4. Subsets of every degree. Every infinite c.e. set A contains a subset of every Turing degree. Choose an infinite computable subset $C = \{c_0 < c_1 < \dots\} \subseteq A$ by the earlier lemma. For an arbitrary set D , put

$$B_D = \{c_n : n \in D\} \subseteq A.$$

An oracle for B_D decides whether $n \in D$ by asking whether $c_n \in B_D$. An oracle for D decides membership of x in B_D by first testing $x \in C$ and, if so, finding its unique index n in the increasing computable enumeration and asking whether $n \in D$. Therefore $B_D \equiv_T D$. This argument treats any one degree at a time; it does not attempt to enumerate the uncountable collection of all degrees.

5. The c.e. degrees. A degree is c.e. if it contains a c.e. set. The greatest c.e. degree is $\mathbf{0}' = \deg_T(K)$, since every c.e. set reduces to K . A c.e. degree can also contain representatives that are not c.e.

Proposition. *Every Turing degree contains exactly countably infinitely many sets. There are $2^{\aleph_0}$ distinct Turing degrees.*

Proof. For a fixed A , every $B \leq_T A$ has characteristic function φ_e^A for some e , so there are at most countably many such B . In particular its degree is countable. The infinitely many distinct sets $A \triangle \{n\}$ are all Turing equivalent to A , giving the lower bound. Since $\mathcal{P}(\mathbb{N})$ has cardinality $2^{\aleph_0}$ and is partitioned into countable classes, the number of classes is $2^{\aleph_0}$ (using the usual cardinal arithmetic with choice). $\square$

There is no maximal Turing degree, as follows from the strictness of the jump proved below.

6.2 Relativization

Working *relative to* A means using A as an oracle. Parametrization relativizes uniformly: for $m, n \geq 1$, there is a total computable injective function s_m^n , independent of the oracle, such that

$$\varphi_{s_m^n(e, \vec{x})}^A(\vec{y}) = \varphi_e^A(\vec{x}, \vec{y}), \quad \vec{x} \in \mathbb{N}^m, \quad \vec{y} \in \mathbb{N}^n.$$

The compiler itself requires no oracle; the program it produces still has access to whichever oracle is supplied. This uniformity lets the basic theorems be applied relative to arbitrary oracles.

A set B is c.e. in A if $B = W_e^A = \text{dom}(\varphi_e^A)$ for some e . Equivalently, an A -oracle procedure enumerates B . The definition makes no inclusion claim between A and B .

The relation “is c.e. in” is not transitive. For example, take $C = \emptyset$, $B = K$, and $D = \overline{K}$. Then B is c.e. in C , and D is even computable in B , hence c.e. in B . But D is not c.e. in C , since otherwise K and its complement would both be c.e. and K would be computable. By contrast, Turing reducibility is transitive.

6.3 The Turing jump

The *Turing jump* of A is

$$A' = \{e : \varphi_e^A(e) \downarrow\}.$$

It is c.e. in A by relative dovetailing, but it is not computable in A : the halting diagonalization works unchanged with all programs given oracle A . Also $A \leq_T A'$: uniformly in n , construct an oracle program that ignores its input and halts just when its oracle answers yes to query n . Its index $h(n)$ satisfies $n \in A \iff h(n) \in A'$. Thus

$$A <_T A'.$$

The jump is the halting problem for oracle- A computations; it is not asserted to be the least degree strictly above A . Monotonicity, $A \leq_T B \implies A' \leq_T B'$, follows by uniformly simulating A -oracle computations with oracle B and asking the resulting halting question of B' . Hence the jump is well defined on degrees. Write $\emptyset^{(0)} = \emptyset$ and $\emptyset^{(n+1)} = (\emptyset^{(n)})'$.

6.4 The arithmetical hierarchy

We use the computability-theoretic normal-form convention: the base classes $\Sigma_0^0 = \Pi_0^0 = \Delta_0^0$ are the computable sets. Define inductively

$$\begin{aligned} B \in \Sigma_{n+1}^0 &\iff x \in B \text{ iff } (\exists y) R(x, y) \text{ for some } R \in \Pi_n^0, \\ B \in \Pi_{n+1}^0 &\iff x \in B \text{ iff } (\forall y) R(x, y) \text{ for some } R \in \Sigma_n^0. \end{aligned}$$

Here tuple coding treats relations as sets, and the equivalences are required for every x . Consecutive quantifiers of the same kind can be grouped into one block. Equivalently, a Σ_n^0 formula has at most n alternating unbounded quantifier blocks over a computable predicate, beginning with an existential block; a Π_n^0 formula begins with a universal block. Define $\Delta_n^0 = \Sigma_n^0 \cap \Pi_n^0$. A set is *arithmetical* if it belongs to one of these classes at some finite level.

In particular,

$$\begin{aligned} B \in \Sigma_1^0 &\iff x \in B \text{ iff } (\exists y) R(x, y) \text{ for a computable } R, \\ B \in \Pi_1^0 &\iff x \in B \text{ iff } (\forall y) R(x, y) \text{ for a computable } R. \end{aligned}$$

The Σ_1^0 sets are exactly the c.e. sets, the Π_1^0 sets are exactly the co-c.e. sets, and Δ_1^0 is the class of computable sets. More generally, $B \in \Sigma_n^0$ if and only if $\overline{B} \in \Pi_n^0$.

Theorem (Strictness). *For every $n \geq 1$ there is a Σ_n^0 set that is not Π_n^0 , and a Π_n^0 set that is not Σ_n^0 .*

Proof. The effective normal forms for formulas give a universal Σ_n^0 relation $S_n \subseteq \mathbb{N}^2$: every Σ_n^0 set is a section $S_{n,e} = \{x : (e, x) \in S_n\}$. This universality is a standard normal-form fact; it is not a claim that all such sections have ordinary c.e. enumerations. Set

$$P_n = \{x : (x, x) \in S_n\}.$$

Then $P_n \in \Sigma_n^0$. If $P_n \in \Pi_n^0$, its complement would be Σ_n^0 , so $\overline{P_n} = S_{n,e}$ for some e . But then

$$e \in P_n \iff (e, e) \in S_n \iff e \in \overline{P_n},$$

a contradiction. Taking complements gives the second assertion. $\square$

Although there are only countably many formulas at each level, their sections need not be c.e. sets. Ordinary effective enumeration of members and syntactic indexing of formulas are different notions.

6.5 Completeness and Post's theorem

Write $B \leq_m A$ if a total computable f satisfies

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

If f can be chosen injective, write $B \leq_1 A$. A set is Σ_n^0 -complete under one-one reductions if it belongs to Σ_n^0 and every Σ_n^0 set is $\leq_1$ it. Define Π_n^0 -completeness in the same way using Π_n^0 sets as the source class. The standard complete sets below have this stronger form of completeness, and therefore also many-one completeness.

Taking complements preserves the same reduction function, so

$$X \text{ is } \Sigma_n^0\text{-complete} \iff \overline{X} \text{ is } \Pi_n^0\text{-complete}.$$

For every $n \geq 1$, $\emptyset^{(n)}$ is Σ_n^0 -complete and $\overline{\emptyset^{(n)}}$ is Π_n^0 -complete.

Theorem (Post). *For $n \geq 0$, the following are equivalent:*

1. $B \in \Sigma_{n+1}^0$.
2. B is c.e. in some Π_n^0 set.
3. B is c.e. in some Σ_n^0 set.
4. B is c.e. in $\emptyset^{(n)}$.

Moreover,

$$B \in \Delta_{n+1}^0 \iff B \leq_T \emptyset^{(n)}.$$

This standard theorem is stated without its full induction proof. For the “some oracle” clauses, an oracle and its complement compute one another, and $\emptyset^{(n)}$ bounds all Σ_n^0 and Π_n^0 sets in Turing degree. The reverse implications follow from the relativized enumeration normal form. The last equivalence follows by applying the first to B and $\overline{B}$ and using the relativized criterion that a set is computable exactly when it and its complement are both c.e.

6.6 Index sets and arithmetical complexity

Set	Definition	Classification
Fin	$\{e : W_e < \infty\}$	Σ_2^0 -complete
Inf	$\{e : W_e = \infty\}$	Π_2^0 -complete
Tot	$\{e : \varphi_e \text{ is total}\}$	Π_2^0 -complete
Con	$\{e : \varphi_e \text{ is total and constant}\}$	Π_2^0 -complete
Rec	$\{e : W_e \text{ is computable}\}$	belongs to Σ_3^0

Here are explicit normal forms and arguments supporting these statements.

$$\begin{aligned}
 e \in \text{Fin} &\iff (\exists s)(\forall t) W_{e,t} \subseteq W_{e,s}, \\
 e \in \text{Inf} &\iff (\forall n)(\exists s) |W_{e,s}| \geq n, \\
 e \in \text{Tot} &\iff (\forall x)(\exists s) \varphi_{e,s}(x) \downarrow, \\
 e \in \text{Con} &\iff (\forall x)(\exists s) [\varphi_{e,s}(x) \downarrow = \varphi_{e,s}(0) \downarrow].
 \end{aligned}$$

The bracket in the last line requires both computations to converge with equal values. All stage-bounded predicates are decidable.

For Π_2^0 -hardness of Tot, write any Π_2^0 predicate as $(\forall x)(\exists s)R(a, x, s)$ with R computable, and compile a into a program which on input x searches for such an s and then returns 0. It is total exactly for the desired a . The same program is total and constant exactly in that case, proving hardness of Con. To reduce Tot to Inf, uniformly enumerate n once all computations $\varphi_e(0), \dots, \varphi_e(n)$ halt. The resulting set is infinite exactly when φ_e is total. Taking complements gives hardness of Fin. Padding makes these index-producing reductions injective.

Finally, a c.e. set is computable if and only if it has a c.e. complement, so

$$e \in \text{Rec} \iff (\exists i) [W_e \cup W_i = \mathbb{N} \text{ and } W_e \cap W_i = \emptyset].$$

More explicitly this is the Σ_3^0 condition

$$(\exists i)(\forall x, s)(\exists t) [x \in W_{e,t} \cup W_{i,t} \text{ and } x \notin W_{e,s} \cap W_{i,s}].$$

The table records this upper bound for Rec; no completeness proof for that set is needed here.

References

- [1] Rebecca Weber, *Computability Theory*, Student Mathematical Library 62, American Mathematical Society, 2012. Publisher's book page.
- [2] Sebastiaan A. Terwijn, *Syllabus Computability Theory*, version 2024. <https://www.math.ru.nl/~terwijn/teaching/syllabus.pdf>. Supplementary reference for the standard results stated without proof.
- [3] Michael Sipser, *Introduction to the Theory of Computation*, 3rd edition, Cengage Learning, 2013. Further background on equivalent Turing machine models and undecidability.

Acknowledgements

This report was prepared as part of the Twoples Reading Program in Mathematics, under the guidance of Dr. Waseet Kazmi. I thank Dr. Kazmi for his guidance throughout the project.