COMPUTATION: DAY 7

BURTON ROSENBERG
UNIVERSITY OF MIAMI

1. Uncomputability

Philosophy seems to me to be amongst men now, in the same manner as corn and wine are said to have been in the world in ancient time. . . . By reasoning I understand computation. — *Thomas Hobbs* [1]

A variant of Lady Lovelace's objection states that a machine can 'never do anything really new'. This may be parried for a moment with the saw, 'There is nothing new under the sun'. Who can be certain that 'original work' that he has done was not simply the growth of the seed planted in him by teaching, or the effect of following well-known general principles. A better variant of the objection says that a machine can never 'take us by surprise'. This statement is a more direct challenge and can be met directly. Machines take me by surprise with great frequency. — *Alan Turing* [2]

My propositions serve as elucidations in the following way: anyone who understands me eventually recognizes them as nonsensical, when he has used them — as steps — to climb beyond them. (He must, so to speak, throw away the ladder after he has climbed up it.) He must transcend these propositions, and then he will see the world aright. — *Ludwig Wittgenstein* [3]

Date: 3 April 2026.

[1]De Corpore, in Thomæ Hobbes Malmesburiensis: Opera Philosophica (Volume 1), William Molesworth (ed.), London: J. Bohn, 1839. ch. 1 sect. 1 and ch.1 sec. 2.

[2]Alan Turing, Computing Machinery and Intelligence, Mind: A quarterly Review of Psychology and Philosophy, Vol. LIX, No. 236. (October 1950)

[3]Wittgenstein, Ludwig. Klement, Kevin C. (ed.). Tractatus Logico-Philosophicus. Proposition 6.54. Translated by Pears, D. F.; McGuinness, B. F. (Side-by-Side-by-Side ed.). University of Massachusetts. Retrieved January 27, 2019. `https://people.umass.edu/klement/tlp/`
In German: *Meine Sätze erläutern dadurch, dass sie der, welcher mich versteht, am Ende als unsinnig erkennt, wenn er durch sie — auf ihnen — über sie hinausgestiegen ist. (Er muss sozusagen die Leiter wegwerfen, nachdem er auf ihr hinaufgestiegen ist.) Er muss diese Sätze überwinden, dann sieht er die Welt richtig.*

2. Universal Turing Machines

2.1 A feature of our computers is that the hardware is fixed and the utility of the machine is determined by software. Even the software is under the control of software, as there is software to download new software onto our computers, install and run the software. This is also a fact about Turing Machines. The state, which is encoded in the transition function δ can be a university program to carry out a program description that is written onto of the tapes before the start of the computation. For any Turing Machine, the δ can be transcribed into a string, which is the program to be run, and the *Universal Turing Machine* can run it.

2.2 A UTM will need a universal set for Σ, Γ and Q. We represent each as a binary code over $\mathcal{B} = \{ 0, 1 \}$. Our universal language for all these elements of a Turing Machine is as it is on our practical computers, strings of 0's and 1's. We consider how to transcribe our generally described TM into one over the language B^*.

2.3 Given the tape alphabet Γ we have an injective map to strings $\mathcal{B}^k$ with k sufficiently large so that $| \Gamma | \leq 2^k$.

2.4 Given the state set Q we have an injective map to string $\mathcal{B}\mathcal{B}^+$ with the preassigned mapping of the start state to 00, the accept state to 01, the reject state to 10 the reject state, and 11 a reserved symbol.

2.5 In representing the map δ as a sting in $\mathcal{B}^*$, the string 0 notates the "move left" action and the string 1 notates the"move right" action.

2.6 A transition is encoded as a string as,

$$(q, \gamma, q', \gamma', a) \implies \mathcal{T} = \mathcal{B}\mathcal{B}^+ \sqcup \mathcal{B}^k \sqcup \mathcal{B}\mathcal{B}^+ \sqcup \mathcal{B}^k \sqcup \mathcal{B}$$

And the entire transition table written on the program tape as,

$$\dashv \left(\sqcup \mathcal{T} \right)^+ \sqcup \sqcup$$

A working tape encoded as,

$$\dashv \left(\sqcup \mathcal{B}^k \right)^* \sqcup \sqcup$$

The current tape head position is over the blank to the left of the k-length tape symbol, and is initialized to the leftmost cell on the tape (which is a blank).

2.7 An additional "current state" tape has the symbol of the current state written on it, and is initialized to the well-known name of the start state 00. After each step this tape checked for equality to 01 or 10, and on match the universal machine enters its own accept or reject state.

2.8 The program that is written into the states of the universal machine is a loop which marches down the program tape looking for a match between the current state and the current input symbol, and on match replaces these with the symbols in the matched transition $\mathcal{T}$. The tape head is advanced according to the last symbol in $\mathcal{T}$. Encountering a double-blank means the table is incomplete and this can be taken as an implied transition to a reject state.

3. Enumeration of Turing Machines

3.1 We have described a UTM, but I wish to go further and describe that machine as $U_i(j)$, a machine on two integer indices, i and j. We will then speak of the computation of the i-th machine on the j-th input. We will have then comprised all possible computations into a single integer function,

$$U_i(j) : \mathbb{Z} \times \mathbb{Z} \to \{ T, F, \perp \}.^4$$

We need to enumerate all possible TM's. Our UTM as a strict syntax for programs, a subset of strings over $\mathcal{B}^*$. This string set is Regular, so we can write a regular expression accepting strings that are TM descriptions, and create a TM that enumerates all strings in $\mathcal{B}^*$ in lexicographic order applying a TM program for the regular expression to decide which strings are programs. As a string matches, it is written on an output tape, and the the sequence of output programs is then our enumeration of programs. If the i-th program is required, this enumerator is run with a counter that stops at the i-th output and writes just that program on a program tape for subsequent use by $U_i(j)$.

3.2 Likewise to find the j-th input. In detail, this is dependent on the program, as the tape symbol set, dependent on k, will be determined by the program syntax.

3.3 However, we now have a program which given i and j, writes the i-th program on the program tape, the j-th input on the work tape and then starts the UTM.

4. The Entscheidungsproblem

Theorem 4.1. The set,

$$A_{TM} = \{ (i, j) \in \mathbb{N} \times \mathbb{N} \mid U_i(j) \}$$

is recursively enumerable but not recursive.

[4] I have hesitated in naming the elements of value set as True or False, and could think Accept or Reject would be better. True and False have common language implications, and they are also the values of the "is an element of" predicate. This investigation is precisely the the difference between how a set is defined in set theory, with values True and False, and how it is defined computationally with Accept and Reject.

4.1 [**Proof**] The set is recursively enumerable because it is recognized by $U_i(j)$.

We prove it is not recursive, suppose A_{TM} were recursive. We will show a contradiction. If A_{TM} were recursive, there is a recursive function $A(i,j)$ which is true when $U_i(j)$ is true, and false otherwise,

$$A(i,j) = \begin{cases} T & U_i(j) = T \\ F & \text{otherwise} \end{cases}$$

The function A resolves the undecided cases of $U_i(j)$. We can then define the recursive function D,

$$D(i) := \neg A(i,i)$$

and construction a machine that includes A, runs it until its result and then interchange Accept and Reject states.

Since D is computable, there is a TM that computes it. Let that be the j-th TM. This machine always halts so we have,

$$D(i) = U_j(i) = A(j,i).$$

We summarize these two definitions of D,

$$D(i) = \neg A(i,i) = A(j,i).$$

We then ask for the result of giving D as input this index,

$$D(j) = \neg A(j,j) = A(j,j).$$

A contradiction. Therefore the set A_{TM} cannot be recursive. □.

Theorem 4.2. The complement set of A_{TM} is not recursively enumerable. That is, you cannot recognize non-halting.

4.2 [**Proof**] If the complement were recursively enumerable, the set A_{TM} (and its complement) would be recursive. □

5. Arithmetic Hierarchy

5.1 We have shown that A_{TM} is not recursive: no Turing machine can decide membership for all inputs, making it *undecidable*. Moreover, its complement is not even recursively enumerable, showing that we have gone beyond the limits of what was previously conceivable for mechanically decidable sets. Thus A_{TM} is also *uncomputable*: no machine can even enumerate all the "yes" instances.

5.2 Both A_{TM} and its complement are undecidable, but they can be distinguished: only A_{TM} is recursively enumerable. An understanding of this distinction is given by

the *arithmetic hierarchy*,[5] where A_{TM} is classified as a Σ_1 set, while its complement is a Π_1 set. Recursive (decidable) sets are classified as Σ_0 or Π_0, since these base classes of the hierarchy coincide.

Definition 5.1. A set $A \subseteq \Sigma^*$ is classified as Σ_1 if there exists a recursive set $A_0 \subseteq (\Sigma^*)^2$ such that

$$x \in A \iff \exists y\ (x, y) \in A_0.$$

A set $A \subseteq \Sigma^*$ is classified as Π_1 if there exists a recursive set $A_0 \subseteq (\Sigma^*)^2$ such that

$$x \in A \iff \forall y\ (x, y) \in A_0.$$

5.3 Because of the logical equality,

$$\neg\Big(\exists x\, \phi(x)\Big) \iff \forall x\, \neg\phi(x)$$

Then the complement of a Σ_1 set is a Π_1 set. For this reason, Π_1 sets are called co-recursively enumerable. This will be true of the entire arithmetic hierarchy.

Lemma 5.1. A set A is Σ_1 if and only if $\overline{A}$ is Π_1.

6. Reductions

Definition 6.1 (Computable function)**.** A *computable function* is and function $f : \Sigma^* \to \Sigma^*$ such that there exists an always-halting Turing machine M that computes the function in the following sense: When M is started with string s on its tape then it runs and halts with string $f(s)$ in its tape.

Definition 6.2 (Reduction)**.** Given $A, B \subseteq \Sigma^*$, a *reduction of A to B* is a computable function $f : \Sigma^* \to \Sigma^*$ that preserves the set inclusion,

(1) $\forall a \in \implies f(a) \in B$,

(2) $\forall a \notin A \implies f(a) \notin B$.

A reduction from A to B is denoted $A \leq_m B$.

Corollary 6.1. If $A \leq_m B$ then $A \setminus \Sigma^* \leq_m B \setminus \Sigma^*$.

Theorem 6.1. Suppose $A \leq_m B$. If B is a recursive set then A is a recursive set. Therefore if A is not recursive then B is not recursive.

[5]The arithmetic hierarchy was defined by Stephen Kleene in *Introduction to Metamathematics*, 1952.

6.1 [Proof] Let f be the reduction map. If B is recursive, then there as a recursive function $g : \Sigma^* \to \{T, F\}$ for which $g(B) = T$ and $g(\overline{B}) = F$. The function $h = g \circ f$ is recursive for which $h(A) = T$ and $h(\overline{A}) = F$. This function is equivalent to a machine deciding A. Hence A is recursive.

The second statement follows by the contrapositive. □

Theorem 6.2. Let M_i be an enumeration of Turing Machines. The language of non-empty languages,

$$\overline{E_{TM}} = \{\, j \in \mathbb{N} \mid \mathcal{L}(M_j) \neq \emptyset \,\}$$

is strictly Σ_1.

6.2 [Proof] The language is recursively enumerable. Consider a variant of M_j such that it inputs (i, t) and runs machine M_j on input i for t steps. If the language is not empty there is an i which for large enough t will accept. This will be found by enumerating of all (i, t) along the diagonal.

We show it is not recursive with a reduction,

$$A_{TM} \leq_m \overline{E_{TM}}$$

as follows. An element $f(i, j)$ maps to a the index l of the machine which acts as,

$$M_l(k) := \Big(M_j(i)\Big) \,?\, T : \bot.$$

(I am using the C language notation for the conditional operator). Such a machine exists — it runs $M_j(i)$ internally and if $M_j(i)$ halts, the machine continues on accepting or non-halting as appropriate. It takes values in A_{TM} to values in $\overline{E_{TM}}$,

$$\begin{aligned} (i, j) \in A_{TM} &\implies M_j(i) = T \\ &\implies \forall k\ M_l(k) = T \\ &\implies \mathcal{L}(M_l) = \Sigma^* \\ &\implies f(i, j) \in \overline{E_{TM}}. \end{aligned}$$

and values not in A_{TM} to values not in $\overline{E_{TM}}$,

$$\begin{aligned} (i, j) \notin A_{TM} &\implies M_j(i) \neq T \\ &\implies \forall k\ M_l(k) = \bot \\ &\implies \mathcal{L}(M_l) = \emptyset \\ &\implies f(i, j) \notin \overline{E_{TM}} \end{aligned}$$

Since A_{TM} is not recursive then $\overline{E_{TM}}$ is not recursive. □

Theorem 6.3. Let M_i be an enumeration of Turing Machines. The language of non-empty languages,

$$E_{TM} = \{\, i \in \mathbb{N} \mid \mathcal{L}(M_i) \neq \emptyset \,\}$$

is of class Π_1 (not recursively enumerable).

6.3 [Proof] Since $\overline{E_{TM}}$ is recursively enumerable, if E_{TM} were recursively enumerable then $\overline{E_{TM}}$ would be recursive. Hence E_{TM} is not recursively enumerable.

It is definable as

$$E_{TM} = \{\, i \in \mathbb{N} \mid \forall j, t\ A_i(j,t) \neq T \,\},$$

where $A_i(j,t)$ is the recursive predicate which simulates M_i on input j for t steps and returns T if M_i accepts within t steps. Thus $A_i(j,t)$ is recursive, and E_{TM} is defined by a universal quantification over a recursive predicate. Hence E_{TM} is of class Π_1. □

7. Equality of Turing machines

7.1 All interesting statements about sets are undecidable. The precise meaning of "interesting" and the proof of this assertion is captured by Rice's Theorem, which we will prove eventually. For the moment, we concern ourselves with a seemingly practical question: given two Turing machines, can we tell if they describe the same or different languages? These questions are not only undecidable, but they occupy a higher level in the arithmetic hierarchy than the statements we have encountered above: one is Σ_2 and the complement statement is Π_2.

7.2 One way to see this is that even if we could somehow eliminate the undecidability of Σ_1 and Π_1, higher-order undecidabilities would still appear. For instance, equality of two Turing machines asserts that for all inputs, the two machines agree; even if we had a predicate that could magically decide halting on any particular input, we are now asking that this agreement be checked for all inputs.

Definition 7.1. The Σ_2 class is of the form,

$$\Sigma_2 = \{\, x \mid \exists r\ \phi(x,r), \text{ where } \phi(x,r) \in \Pi_1 \,\}$$

The Π_2 class is of the form,

$$\Pi_2 = \{\, x \mid \forall r\ \phi(x,r), \text{ where } \phi(x,r) \in \Sigma_1 \,\}$$

Lemma 7.1. If A is Σ_2 and there is a non-recursive Π_1 set B such that $B \leq_m A$, then A is not Σ_1.

7.3 [Proof] Suppose $A \in \Sigma_1$. The reduction $B \leq_m A$ shows that B can be decided by at Σ_1 machine. Hence $B \in \Sigma_1$ as well as Π_1. Hence it is recursive, a contradiction. □

Theorem 7.1. The set describing non-equality of Turing machines,

$$\overline{EQ_{TM}} = \{\, (i,j) \in \mathbb{N} \times \mathbb{N} \mid M_i \neq M_j \,\}$$

is strictly Σ_2.

7.4 [Proof] A description of $\overline{EQ_{TM}}$, is,

$$(i,j) \in \overline{EQ_{TM}} \iff \exists x \left[(x \in \mathcal{L}(M_i) \wedge x \notin \mathcal{L}(M_j)) \ \vee \ (x \notin \mathcal{L}(M_i) \wedge x \in \mathcal{L}(M_j)) \right].$$

Crucially, the statement $x \notin \mathcal{L}(M)$ is Π_1. Hence $\overline{EQ_{TM}}$ is Σ_2.

We show it is not Σ_2 by the two reductions,

$$\overline{A_{TM}}, A_{TM} \leq_m \overline{EQ_{TM}}.$$

Recall the map $f(i,j)$ maps to a the index l of the machine which acts as,

$$M_l(k) := \Big(M_j(i) \Big) \, ? \, T : \bot.$$

and consider the map

$$h(i,j) \mapsto (f(i,j), M_{\Sigma^*})$$

where M_{Σ^*} is the machine that accepts everything. It preserves truth from acceptance to equality,

$$\begin{array}{rcl}
(i,j) \in \overline{A_{TM}} & \implies & M_j(i) \neq T \\
& \implies & \forall k,\ M_l(k) = \bot \\
& \implies & (f(i,j), M_{\Sigma^*}) \in \overline{EQ_{TM}}. \\
(i,j) \in A_{TM} & \implies & M_j(i) = T \\
& \implies & \forall k,\ M_l(k) = T \\
& \implies & (f(i,j), M_{\Sigma^*}) \in EQ_{TM}.
\end{array}$$

This gives us the reductions,

$$\overline{A_{TM}} \leq_m \overline{EQ_{TM}}.$$

Consider now the map

$$h(i,j) \mapsto (f(i,j), M_{M_\emptyset})$$

where $M_\emptyset$ is any Turing machine that accepts nothing. It preserves truth from acceptance to non-equality.

$$\begin{aligned}
(i,j) \in A_{TM} &\implies M_j(i) = T \\
&\implies \forall k,\ M_l(k) = T \\
&\implies (f(i,j), M_\emptyset) \in \overline{EQ_{TM}}. \\
(i,j) \notin A_{TM} &\implies M_j(i) \neq T \\
&\implies \forall k,\ M_l(k) = \bot \\
&\implies (f(i,j), M_\emptyset) \notin \overline{EQ_{TM}}.
\end{aligned}$$

This gives us the reductions,

$$A_{TM} \leq_m \overline{EQ_{TM}}.$$

This gives the result. □

Theorem 7.2. The set describing equality of Turing machines,

$$EQ_{TM} = \{\, (i,j) \in \mathbb{N} \times \mathbb{N} \mid M_i = M_j \,\}$$

is strictly Π_2.

7.5 [**Proof**] A description of EQ_{TM}, is,

$$(i,j) \in EQ_{TM} \iff \forall x \left[(x \in \mathcal{L}(M_i) \wedge x \in \mathcal{L}(M_j)) \ \vee \ (x \notin \mathcal{L}(M_i) \wedge x \notin \mathcal{L}(M_j)) \right].$$

Crucially, the statement $x \in \mathcal{L}(M)$ is Σ_1. Hence EQ_{TM} is Π_2.

Referring back to the proof of the previous theorem, we negate the reductions and have,

$$\overline{A_{TM}}, A_{TM} \leq_m EQ_{TM}.$$

This gives EQ_{TM} is Π_2. □

8. Rice's Theorem

8.1 Rice's theorem states that any non-trivial property of a language is undecidable. The proof is s generalization of the reduction of the previous section.

Theorem 8.1 (Rice's Theorem)**.** Any non-trivial property of recursively enumerable sets is undecidable.

8.2 [**Proof**] A non-trivial property is a property that some languages have and some languages do not have. Arrange for the language of its complement to be such that

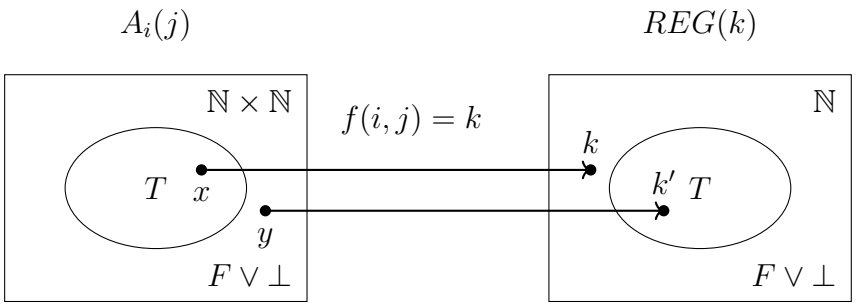

FIGURE 1. Reduction $A_i(j) \leq_m REG(k)$

machine P recognizes a language with the property and the empty language does not have the property. Then the construction,

$$M_{f(i,j)}(k) := \Big(A_j(i)\Big) \,?\, P(k) : \perp$$

is a reduction $A_{TM} \leq_m P$. Therefore P is undecidable. □

8.3 We give an example of Rice's theorem, with an explicit construction to show that the set REG all Turing machines whose language is regular is undecidable. In fact $REG \in \Sigma_3$.

Theorem 8.2. Let M_i be an enumeration of Turing machines. The language of regular languages is not recursively enumerable.

8.4 [Proof] Let N be a machine accepting a non-regular language,

$$\mathcal{L}(N) = \{\, 0^i\, 1^i \,\}.$$

Consider the map $f(i,j)$ which maps the index of a Turing machine i and test input j to the index of a Turing machine acting as,

$$M_{f(i,j)}(k) := \Big(A_i(j)\Big) \,?\, N(k) : \perp.$$

The resulting machine accepts a non-regular language if $A_i(j) = T$ and otherwise accepts a regular language. So

$$\overline{A_{TM}} \leq_m REG.$$

Therefore REG is not recursively enumerable.

The reduction is diagrammed in Figure 1. The point x is an (i,j) pair such that $A_i(j) = T$. It maps to the index k of a Turing machine that accepts a non-regular language (specifically $0^n 1^n$). The point y is an (i,j) pair such that $A_i(j) \neq T$ maps

to the index k' of a Turing machine that accepts a regular language (specifically the empty language). □

8.5 Let N_i be a enumeration of DFA's, Then the language REG is,

$$REG = \{ j \mid \exists\, i,\ \mathcal{L}(N_i) = \mathcal{L}(M_j) \}$$

While we have EQ, equality of Turing machines, we could show that equality between a Turing machine and a DFA is also Π_2. Hence $REG \in \Sigma_3$.