COMPUTATION: DAY 6

BURTON ROSENBERG
UNIVERSITY OF MIAMI

Contents

1. The Arithmetic Hierarchy 1
2. Undecidability 3
3. Rice's Theorem 5
3.1. Equality of Turing machines 6
4. Decidability of Promise Problems 7
4.1. Regular Languages 8
4.2. Context Free Languages 8
5. Church-Turing Thesis 9
6. First order logic 10

1. The Arithmetic Hierarchy

On the previous day we discovered that not all functions are recursive. In fact, the function $A_i(j)$ defines a set of true instances but not a set of not-true instances because the not-true instances include non-halting cases. Taking it from the point of view of actually computing $A_i(j)$, while the machine continues to run we cannot say if in the future it will eventually accept or reject the input.

We then inferred that the complement of this set cannot even be recursively enumerable. Hence A_{TM} is *undecidable* — since recursively enumerable languages include recursive languages, the set A_{TM} is strictly recursively enumerable.

Many other sets are undecidable, and many such sets are of practical interest. For instance, given two Turing machines, are they two different implementations of the same function, or are they dissimilar. This problem is undecidable. We day will prove *Rice's Theorem* that says that all interesting properties of languages are undecidable.

We will also show that for some languages, neither it nor its complement are recursively enumerable, defining sets of a "higher" undecidability.

Date: 2 April 2025.

Definition 1.1. Let $A(a, n)$ be a recursive function. Then the language defined as,

$$a \in A \iff \exists\, n\ A(a, n) = T$$

is called a Σ_1 language; and the language defined as,

$$a \in A \iff \forall\, n\ A(a, n) \neq T$$

is called a Π_1 language. Language classes $\Sigma_0 = \Pi_0$ are also defined, and are both the recursive languages.

The class Σ_1 is exactly the class of recursively enumerable functions. We create a machine that is certainly halting but stopping it after a given number of steps. And then search over the number of steps for a true result, if that ever occurs.

Definition 1.2. A step-bounded Turing machine is a machine $A(j; t)$ that decides halts after no more than t computation steps with the result T when j is accepted, F when j is rejected, and $\perp$ when j is neither accepted nor rejected by the t-th step. The language of $A(j)$ is,

$$A(j) = \lim_{n\to\infty} A(j; t)$$

Theorem 1.1. A language is recursively enumerable if and only it is Σ_1.

Proof: Give a Turing machine $A(a)$ with the step-bounded version $A(a; t)$, It is defined by

$$a \in A \iff \exists t \in \mathbb{N}\, A(a; t) = T.$$

□

Theorem 1.2. Let M_i be an enumeration of Turing Machines. The language of non-empty languages,

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

is recursively enumerable.

Proof: Enumerate $\mathbb{N} \times N$ along the diagonal,

$$d = \langle\, (0,0), (1,0), (0,1), (2,0), (1,1), (0,2), (3,0), \ldots \rangle.$$

Then

$$i \in \overline{E_{TM}} \iff \exists\, n \text{ such that } d(n) = (j, t) \text{ and } A_i(j; t) = T.$$

□

Theorem 1.3. Both $\overline{A_{TM}}$ and E_{TM} are Π_1.

Proof: Note that,

$$(i,j) \in \overline{A_{TM}} \iff \forall t\ A_i(j;t) \neq T.$$

For E_{TM} one diagonalizes over inputs and time bounds,

$$i \in E_{TM} \iff \forall n, A_i(j;t) \neq T \text{ where } d(n) = (j,t).$$

□

2. Undecidability

A language which is not decidable is called undecidable. That is an undecidable language is not in Σ_0. We have already shown that

$$A_{TM} = A_i(j)$$

the universal acceptance language is in Σ_1 but not Σ_0, hence it is said to be an undecidable language. We can show other languages to be undecidable using the notion of a reduction (by a recursive function) to establish precedence in the arithmetic hierarchy.

Definition 2.1. A *recursive 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 2.2. Given $A, B \subseteq \Sigma^*$, a *reduction of A to B* is a recursive 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 $A \leq_m B$.

Theorem 2.1. If $A \leq_m B$ then $\overline{A} \leq_m \overline{B}$.

Theorem 2.2. 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.

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 2.3. Suppose $A \leq_m B$. If B is a recursively enumerable set then A is a recursively enumerable set. Therefore if A is not recursively enumerable, B is not recursively enumerable.

Proof: Similar to the above proof.

Theorem 2.4. Let M_i be an enumeration of Turing Machines. The language of non-empty languages,

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

is not recursive.

Proof: 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,

$$M_{f(i,j)}(k) \equiv \text{if } A_i(j) \text{ then } T \text{ else } \perp$$

That is, the function f writes the instructions for a Turing machine that uses a universal Turing machine to compute $A_i(j)$. If the result is accept, the machine will immediately accept its own input k. If the result is reject, the machine enters a loop and does not halt. The function f then finds the index of this new Turing machine on the list of Turing machines and writes this value on the tape. This is a reduction

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

It takes values in A_{TM} to values in $\overline{E_{TM}}$,

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

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

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

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

Theorem 2.5. 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 not recursively enumerable. It is of class Π_1.

Proof: Since $\overline{E_{TM}}$ is recursively enumerable, if E_{TM} were recursively enumerable $\overline{E_{TM}}$ would be recursive. It is definable as,

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

hence it is of class Π_1. □

3. Rice's Theorem

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. As another example of this reduction, consider the set REG all Turing machines whose language is regular.

Theorem 3.1. Let M_i be an enumeration of Turing machines. The language of regular languages is not recursively enumerable.

Proof: Let N be a machine accepting a non-regular language, such as $\mathcal{L}(N) = \{\, 0^i\, 1^i \,\}$. 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,

$$M_{f(i,j)}(k) \equiv \text{if } A_i(j) \text{ then } N(k) \text{ else } \perp.$$

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

$$\begin{aligned}
(i,j) \in A_{TM} &\implies A_i(j) = T \\
&\implies M_{f(i,j)}(k) = N(k) \\
&\implies \mathcal{L}(M_{f(i,j)}) = \{\, 0^i\, 1^i \,\} \\
&\implies f(i,j) \in \overline{REG}. \\
(i,j) \notin A_{TM} &\implies A_i(j) \neq T \\
&\implies \forall k\ M_{f(i,j)}(k) = \perp \\
&\implies \mathcal{L}(M_{f(i,j)}) = \emptyset \\
&\implies f(i,j) \in REG
\end{aligned}$$

So

$$\overline{A}_{TM} \leq_m REG.$$

Therefore REG is not recursively enumerable. □

Definition 3.1. If a set is definable as

$$G = \{\, i \in \mathbb{N} \,|\, \exists x\, \forall y, F(x,y,i) = T \,\}$$

for an F of class Σ_n, then G is of class Σ_{n+2}.

Theorem 3.2. The language of regular languages is in Σ_3.

Proof: Let M_i be an enumeration of Turing machines. Let D_j be an enumeration of all Finite Automata. Let $N_j(k)$ a universal machine that simulates D_j on k. The language of regular languages is definable as,

$$REG = \{\, i \in \mathbb{N} \mid \exists j \, \forall k \, \exists t, A_i(k;t) = N_j(k) \,\}.$$

The predicate

$$A_i(k;t) = N_j(k)$$

is recursive. Therefore

$$\exists t, A_i(k;t) = N_j(k)$$

is of type Σ_1, and

$$\exists j \, \forall k \, \exists t, A_i(k;t) = N_j(k)$$

is of type Σ_3. □

Theorem 3.3 (Rice's Theorem)**.** Any non-trivial property of recursively enumerable sets is undecidable.

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

$$M_{f(i,j)}(k) \equiv \text{if } A_i(j) \text{ then } P(k) \text{ else } \perp$$

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

3.1. **Equality of Turing machines.**

Theorem 3.4. Equality and non-equality for Turing machines are both non-recursively enumerable.

Proof: Non-equality is a Σ_2 statement,

$$\overline{EQ_{TM}} = \{\, (i,j) \in \mathbb{N} \times \mathbb{M} \mid \exists \, k, t_k \, \forall \, t, A_i(k;t) \neq A_j(k;t) \vee t < t_k \,\}$$

and equality is a Π_2 statement,

$$EQ_{TM} = \{\, (i,j) \in \mathbb{N} \times \mathbb{M} \mid \forall \, k, t \, \exists \, t_k, A_i(k;t_k) = A_j(k;t_k) \wedge t_k \geq t \,\}$$

Consider the map $h_1(i,j) \mapsto (f(i,j), M_\emptyset)$ where,

$$M_{f(i,j)}(k) \equiv \text{if } A_i(j) \text{ then } T \text{ else } \perp$$

and $M_\emptyset$ is any Turing machine that accepts nothing. Previously it was described how f is a recursive function. It preserves truth from acceptance to non-equality.

$$\begin{aligned}
(i,j) \in A_{TM} &\implies A_i(j) = T \\
&\implies \mathcal{L}(M_{f(i,j)}) = \Sigma^* \\
&\implies M_{f(i,j)} \neq M_\emptyset \\
&\implies (f(i,j), M_\emptyset) \in \neg EQ_{TM}. \\
(i,j) \notin A_{TM} &\implies A_i(j) \neq T \\
&\implies \mathcal{L}(M_{f(i,j)}) = \emptyset \\
&\implies M_{f(i,j)} = M_\emptyset \\
&\implies (f(i,j), M_\emptyset) \notin \neg EQ_{TM}
\end{aligned}$$

Consider the very similar map $h_2(i,j) \mapsto (f(i,j), M_{\Sigma^*})$ where M_{Σ^*} is the machine that accepts everything. It preserves truth from acceptance to equality,

$$\begin{aligned}
(i,j) \in A_{TM} &\implies M_i(j) = T \\
&\implies (f(i,j), M_{\Sigma^*}) \in EQ_{TM}. \\
(i,j) \notin A_{TM} &\implies M_i(j) \neq T \\
&\implies (f(i,j), M_{\Sigma^*}) \notin EQ_{TM}
\end{aligned}$$

So we have two reductions,

$$A_{TM} \leq_m \neg EQ_{TM}, \;\; A_{TM} \leq_m EQ_{TM}$$

so neither can be recursive. As reductions are stable by complementing both sides, we also have,

$$\neg A_{TM} \leq_m EQ_{TM}, \;\; \neg A_{TM} \leq_m \neg EQ_{TM}$$

So neither can be recursively enumerable either.

These are sets that cannot be decided, and further neither the set nor its complement can be recognized.

4. Decidability of Promise Problems

If we know that a language is Regular or Context Free, some properties of these languages are decidable. We can know that the language is these classes because they are presented in particular ways. For instance, if we are given a finite automata, we know the language is regular. If we are given a Context Free Grammar, we know the language is a Context Free Language.

Another way to describe the situation is that the index is of a Turing machine, and we are promised that the Turing machine accepts are Regular (or Context Free) language. It's a bit non-constructive, it could happen that the Turing machine is a

Finite Automata simulator with a finite automata program baked right in. However I mention it here so that these results have a wider context.

4.1. **Regular Languages.** While the set REG is undecidable, many properties of regular languages are decidable.

Theorem 4.1. It is decidable if a regular language is empty, or contains all strings; and if two regular languages are equal.

Proof: Given a regular language there is a finite automata accepting that language. Working backwards from accept states will provide a path, if any, for some accepted string. This decides emptiness.

The language accepts all strings if the complement is empty. Given the finite automata for the language, exchange the accept and non-accept states and determine if the resulting language is empty.

Given two regular languages there are two finite automata F_A and F_B accepting the respective languages. A finite automata can be constructed that accepts

$$F_A = F_B \iff (F_A \wedge \neg F_B) \vee (\neg F_A \wedge F_B) = \emptyset$$

and then decide the emptiness of this regular language. The first term is the empty set when $F_A \subseteq F_B$, and the second term is the empty set when $F_B \subseteq F_A$. For the sum to be empty, both these conditions need to be true. □

4.2. **Context Free Languages.** The same set of questions for Context Free Languages is more difficult.

Theorem 4.2. Non-equality for Context Free Languages is recursively enumerable.

Proof: A non-deterministic Turing machine can guess the string on one language and not the other. Determining membership is recursive. If we enumerate CFG's as G_i, then the language is,

$$\neg EQ_{CFL} = \{\, i, j \mid \exists s, G_i(s) \neq G_j(s) \,\}$$

and the complement is co-recursively enumerable,

$$EQ_{CFL} = \{\, i, j \mid \forall s, G_i(s) = G_j(s) \,\}$$

Since the G_i is recursive, respectively these languages are class Σ_1 and Π_1. □

Theorem 4.3. Non-equality for Context Free Languages is undecidable.

Theorem 4.4. Emptiness for Context Free Languages is decidable.

Proof: The the grammar in Chomsky Normal Form, apply a marking algorithm that determines for each variable whether that variable is *productive*, that is can become by the grammar a string of terminals. If the rule $S \to \varepsilon$ is present the language is

non-empty. Mark any variable V if it appears in the rule $V \to v$, with v a terminal. Mark any variable V if it appears in the rule $V \to A\,B$, where A and B are both marked. Continue until no more markings can be made. The language is non-empty if and only if S is marked. □

Theorem 4.5. Whether a Context Free Languages is all strings is undecidable, and Π_1, as it is the predicate for the i-th language G_i, $\forall s, G_i(s)$ with $G_i(s)$ recursive.

5. Church-Turing Thesis

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]

6.54 *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.*

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.

[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)

— *Ludwig Wittgenstein* [3]

6. First order logic

The classes Σ_i and Π_i were defined using the logical operators "for all" and "there exists". This section reviews the logic of these symbols.

A *predicate* is a function in boolean algebra. It may have variables that are connected with the logical operations of and, or and not. Other operations are introduced, such as implication, however they can be defined in terms of and, or and not. For instance,

$$(x \implies y) \iff ((\neg x) \vee y)$$

A logical formula is expressed with *unbound variables*. For instance, the formula $\phi(x, y)$ has two unbound variables, x and y. The *quantifiers* $\forall$ and $\exists$ can bind the variable in a formula, as follows,

$$\exists\, x\ \phi(x, y) \text{ or } \forall\, y\ \phi(x, y)$$

The result is a formula with one unbound variable.

While a formula can be written with the quantifiers mixed inside the formula. e.g $x \wedge (\forall\, y\ y \vee z)$, the quantifiers can alway be brought to the front of the formula, putting the formula in *prenex form*,

$$x \wedge \big(\forall\, y(y \vee z)\big) \iff \forall\, y\big(x \wedge (y \vee z)\big)$$

In this way, a quantified boolean formula is always an alternation of quantifies binding variables in a predicate, perhaps leaving some variables free. If no variables are free then the formula is either true or false, for instance,

$$\forall\, a, b\ \exists\, x(a + bx = 0).$$

is true when the range of discourse is the rational number. It is false when the range of discourse is the integers.

Two important rules for quantifiers are,

$$\begin{aligned} \neg\big(\exists\, x\ \phi(x, y)\big) &\iff \forall\, x\ \neg\phi(x, y) \\ \neg\big(\forall\, x\ \phi(x, y)\big) &\iff \exists\, x\ \neg\phi(x, y) \end{aligned}$$

[3]Wittgenstein, Ludwig. Klement, Kevin C. (ed.). Tractatus Logico-Philosophicus. 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/`