UNDECIDABILITY AND THE STRUCTURE OF THE TURING DEGREES

PHILIP ADAMS

Abstract. This paper explores the structure and properties of the Turing degrees, or degrees of undecidability. We introduce the Turing machine, an abstract model of computation, in order to develop the concepts of undecidability and Turing reduction. We demonstrate the technique of proof by reduction through a series of examples of undecidable problems related to context-free grammars. We then employ reducibility to consider a partial ordering on the set of Turing degrees, $\mathcal{D}$. Finally, we prove a variety of theorems related to the structure of $\mathcal{D}$.

Contents

1. Introduction

The field of computability theory focuses on abstract models of computation and the sorts of problems that they are able to solve. Unlike complexity theory, which is focused on the resources required to perform a computation, computability theory focuses only on whether a certain computation is possible on a given machine. The practical application of the theory focuses on models of computation of equivalent power to general purpose computers, allowing for it to be determined whether it possible to solve specific problems on actual machines. The Church-Turing thesis, a significant conjecture in the field, states that the model that we use to understand

Date: October 1, 2018.

the power of general purpose computers has the same power as humans applying algorithms. If true, it means that it is not possible to design a computational process that is able to solve more problems than a general-purpose computer. However, even if it is true, it is still possible to imagine models that have more power than general purpose computers by endowing them with *oracles* (see Definition 5.4) which have the ability to evaluate problems that we know general-purpose computers cannot. Because of this, computability is still able to draw distinctions in the difficulty of problems too difficult for general-purpose computers to solve, dividing them into *Turing degrees*, or *degrees of unsolvability.* Additionally, it is possible to construct models of computation with less power than general-purpose computers, and thus classify the difficulty of problems that general-purpose computers can solve. We can also show the differences in difficulty of problems simply by showing that it is possible to use the answer of one problem to solve another, a technique called *reduction.* So, it is possible to create an infinite hierarchy describing the difficulty of problems, and to investigate the sort of structure this hierarchy takes.

1.1. **Decision Problems.** What does it mean to *solve a problem*? What is a *problem*? These questions may seem esoteric, but they must be addressed in order to build a well defined theory of the difficulty and computability of problems. In this paper, we discuss a specific type of problem, the *decision problem.* Decision problems are essentially sets of natural numbers, and to solve the problem is to be able to decide whether any given element is inside the set or not. Because set theory is commonly used as the foundational system of mathematics, we can convert problems expressed in terms of more complex mathematical objects into decision problems, and thus by studying decision problems can build a system that is able to assess the computability of all mathematical problems. For example, we can express strings over the alphabet $\{0, 1\}$ as sets of natural numbers, and thus can study problems about languages of such strings with the same tools we use for decision problems. Going further, we can express the source code of a computer program as a string over the alphabet $\{0, 1\}$, and so can consider questions about the behavior of computer programs.

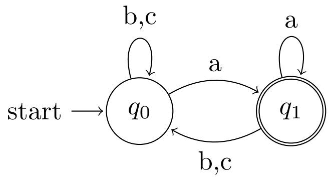
1.2. **Overview.** This paper aims to develop the ideas presented in the introduction in a natural manner, and at a level of rigor accessible to an interested undergraduate. We begin by presenting a few simple models of computation and assessing their ability to compute various sets. We then introduce the Turing Machine, and mention alternative models of equivalent power. Next, we introduce the concepts of undecidability and Turing reducibility, and present the halting problem as an example of an undecidable problem. In the interest of developing a strong intuition regarding the concept of undecidability, we examine a variety of decidable and undecidable problems about Context-Free Grammars. Finally, we employ the Turing reduction to introduce a structure on the set of decision problems, and present some basic results about its properties.

2. Models of Computation

Upon consideration of different types of problems, it quickly becomes clear that some are much more difficult than others. For example, when considering words over the alphabet $\{a, b, c\}$, it seems much easier to determine whether a word ends in an a than whether a word is a palindrome. It seems even harder still to determine whether the word is of the form $a^nb^nc^n$. In order to formalize these notions of difficulty, we need to build abstract models of computation, and then test their ability to decide such questions. We observe that a machine that can solve the first problem only needs to "remember" the same amount of information no matter how long the input is: the last letter of the word. In contrast, the amount of memory that the second and third problems require is dependent on the size of the input, because the first half of the palindrome or n, respectively, can be arbitrarily large. The difference between the second and third problems is more subtle: the second problem can be solved moving only one direction in memory, while the third problem requires the ability to "look back" in memory.

Based on these differences, we can begin to construct different models of computation that have just enough power to solve each problem. The first problem can be solved by a *Finite State Automaton (FSA)*, a collection of a finite number of states, a transition function, and a set of accepting states. The FSA accepts the input if the sequential application of each element to the input finishes in an accepting state, and rejects otherwise. Finite State Automata are often represented by diagrams as in Figure 1, a representation of an FSA that decides the first problem. An explanation of the conventions used in these diagrams is available in Sipser's *Introduction to the Theory of Computation* [9, pp. 34, 114]

FIGURE 1. An FSA that determines whether a string ends in 'a'.

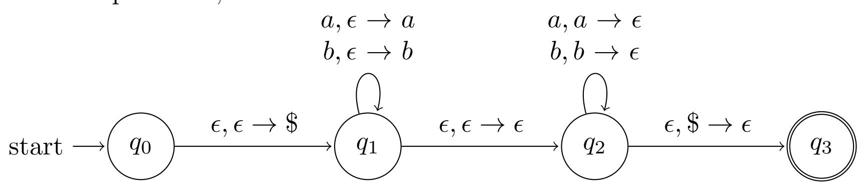


The second problem can be solved by a machine called a *Pushdown Automaton (PDA)*, which is essentially a finite state automaton with the addition of a stack. Rather than transforming the current state and the input into a new state, as with a finite state automata, the transition function of a PDA also considers the stack, which it can pop and push from. Within the collection of PDAs, there is another division between *Deterministic Pushdown Automata (DPDAs)* and *Nondeterministic Pushdown Automata*. For DPDAs, the transition function only outputs one move, rather than a set of moves. Nondeterministic PDAs are able to recognize more languages than DPDAs. Figure 2 is a graphical representation of a PDA that decides palindromes. In this representation, ϵ is an empty input, and \$ is a special symbol that denotes the bottom of the stack. The moves are represented by a double of input and stack operation.

2.1. **Turing Machines.**

Finally, we arrive at the Turing Machine (TM), devised by Alan Turing in his paper *On Computable Numbers, With an Application To the Entscheidungsproblem* [12]. Turing Machines offer an unrestricted model of computation, equivalent to the general purpose computers that we are familiar with today. This equivalence is expressed by a pair of concepts called *Turing completeness* and *Turing equivalence*. A system of computation is Turing complete if it is able to simulate an arbitrary Turing machine, and is Turing Equivalent if there is a Turing

Figure 2. A Pushdown Automaton that decides palindromes over the alphabet a, b



machine that can simulate it. Essentially, Turing completeness is being *at least* as powerful as Turing machines, while Turing equivalence is being *exactly* as powerful as a Turing machine. Turing machines are themselves Turing complete, as are other abstract models of computation such as the *lambda calculus*. General purpose programming languages are also considered Turing complete, because although they are run on real machines with finite storage space, this space is theoretically unbounded, and thus the expressive power of the languages is unbounded even if any particular instance of a program run on an actual machine would not be able to simulate all Turing machines.

A basic informal definition of a Turing machine is a collection of an infinite tape of discrete cells, a head that can move back and forth between cells on the tape as well as read and write from the tape, a register that stores the current state of the machine out of a finite set of possible states and starts in a start state, a set of accepting states, and a transition function that at each step allows tells the machine whether to write to the tape, move the head, or change states. There are modifications to this definition which allow for multiple tapes or nondeterminism, but they are all equivalent to the basic definition. Additionally, a PDA with two stacks is equivalent to a Turing machine, as the stacks can be thought of as extending out in opposite directions, forming an infinite tape.

3. Undecidability

In the previous section we were concerned primarily with problems that our models of computation were able to solve. However, much as an FSA is unable to solve the problem of whether a string is a palindrome, there are problems that are too difficult for even Turing Machines to solve. When a decision problem A cannot be solved by a model $\mathcal{M}$, we say that A is undecidable for $\mathcal{M}$. When we just say that A is undecidable, we mean that it is undecidable for TMs.

It can be difficult to come up with obvious examples of problems that are undecidable. After all, we usually talk about sets based on the patterns they exhibit, and those patterns are typically simple enough to be enumerated by a computer. A promising place to look for problems that a model cannot decide is problems about the model itself, and the most famous undecidable problem is of that type. The Halting Problem, introduced by Alan Turing in the same paper where he introduced the Turing machine, asks whether it is possible to contsruct a Turing machine that can determine whether a Turing machine run on a given input will ever halt and return an answer [12]. The proof is through a method called *diagonalizdation*, due to its similarity to Cantor's diagonal argument.

Problem 3.1 (The Halting Problem)**.** *Let M be a Turing Machine and i be an input. If M is run on i, will it eventually halt?*

Proof of Undecidability. Suppose for the sake of contradiction that the halting problem is decidable. Then, there exists some machine $\mathcal{O}$ that can decide the halting problem. Then, we can construct a Turing machine H that simulates $\mathcal{O}$ on its input (M, i). In the case where $\mathcal{O}(M, i)$ accepts, H enters into an infinite loop, whereas in the case where $\mathcal{O}(M, i)$ does not accept, H halts. Consider $H(H, \epsilon)$. If H halts, then $\mathcal{O}(H, \epsilon)$ accepts, and so H enters an infinite loop, and so it does not halt. But if H does not halt, then $\mathcal{O}(H, \epsilon)$ does not accept, so H halts. This is a contradiction, so our premise that the halting problem is decidable must be false. □

3.1. **Reducibility.** While it is possible to prove that a problem is undecidable directly, as in Problem 3.1, it is often more convenient to prove undecidability through comparison to problems which are already known to be undecidable. This comparison takes place through the technique of Turing reduction.

Definition 3.2. Let A and B be decision problems. We say that A is *Turing reducible* to B and write $A \leq_T B$ if any instance of problem A can be "converted" by a Turing Machine into an instance of problem B. More formally, $A \leq_T B$ if it is possible to construct an oracle machine for B that decides A (see Definition 5.4).

So, we have that if A is reducible to B, then A can be no harder than B, because any solution to B also leads to a solution of B. So, if we have some problem B that we would like to prove is undecidable, we can do so by showing that some problem A which is already known to be undecidable is reducible to B. Since A cannot be harder than B, it follows that B must also be undecidable. Alternatively, if we would like to show that some problem A is decidable, it is sufficient to show that it is reducible to some problem B that is known to be decidable [9, 6].

4. Context Free Languages

Definition 4.1. A Context-Free Grammar (CFG) G is a finite set of nonterminal symbols, a finite set of terminal symbols, finite number of production rules that produce a string of terminal and nonterminal symbols from a nonterminal symbol, and a starting symbol.

Definition 4.2. The *language* $L(G)$ of a CFG G is the set of words that can be formed by the successive application of production rules to the start symbol. These languages are called Context-Free Languages (CFLs).

Remark 4.3. The set of terminal symbols of a CFG G is usually denoted by Σ, and is called the *alphabet* of $L(G)$.

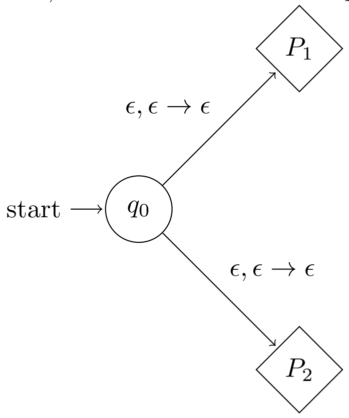
CFGs provide another useful model of computation. However, the languages that they are able to express are exactly the same set of languages as PDAs [3]. This equivalence allows us to choose between the two models and use the one that is most useful for proving facts about these languages.

Theorem 4.4 (Closure Properties of CFLs)**.** *Context-Free Grammars have a variety of closure properties:*

- *Let L_1, L_2 be CFLs. Then $L_1 \cup L_2$ is a CFL.*

Proof. The languages L_1 and L_2 are Context-Free Languages, so there exist Pushdown Automata P_1 and P_2 that decide them. Consider the PDA U shown in Figure 3, which simulates P_1 and P_2 on the input. U is a PDA that decides $L_1 \cup L_2$, so it follows that $L_1 \cup L_2$ is a CFL. □

FIGURE 3. U, a PDA that simulates P_1 and P_2.


P_1
$\epsilon, \epsilon \to \epsilon$
start
q_0
$\epsilon, \epsilon \to \epsilon$
P_2

- *Let L_1, L_2 be CFLs. Then the concatenation*

$$\{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$$

  is a CFL.

  Proof. The languages L_1 and L_2 are Context-Free Languages, so there exist CFGs G_1, G_2 with start symbols S_1, S_2 that produce them. Suppose, without loss of generality, that G_1, G_2 have no nonterminal symbols in common (if they do, then we can rename them). Consider the grammar $S \to S_1 S_2$. Then, this is a context free grammar, and its language is the concatenation of L_1 and L_2. So, the concatenation of L_1 and L_2 is a CFL. □

- *Let L be a CFL. Then,*

$$L^* = \bigcup_{i \geq 0} \{w_1 \dots w_k \dots w_i \mid w_k \in L\}$$

  is a CFL.

  Proof. The language L is a CFL, so there exist CFGs G with start symbol S_1 that produces it. Consider the grammar given by $S \to S_1 S \mid \epsilon$. This is a CFG that produces L^*, so it follows that L^* is a CFL. □

- *Intersection with regular languages [3].*

We have that the emptiness and finiteness problems are decidable for CFGs [3, 9]. Another problem that is decidable for CFGs is the question of whether a given CFL is wholly contained by a given regular language.

Problem 4.5 (Regular Containment)**.** *Take some context-free grammar G and some regular language R. Is $L(G) \subseteq R$?*

Proof of Decidability. We have that $L(G) \subseteq R$ if and only if $\overline{L(G)} \cup R = \Sigma^*$. So, it follows that $L(G)$ is contained by R if and only iff $\overline{\overline{L(G)} \cup R} = \varnothing$. By DeMorgan's laws, we have that this is equivalent to the statement $L(G) \cap \overline{R} = \varnothing$. Regular languages are closed under complement and context-free languages are closed under intersection with a regular language, so it follows that $L(G) \cap \overline{R}$ is a context free language. So the problem of containment by a regular language is reducible to the problem of emptiness, which we know to be decidable. So, the problem of containment by a regular language is decidable. □

4.1. **Undecidable Problems.**

Problem 4.6 (The Post Correspondence Problem). *Consider some alphabet Σ and two finite lists of words over Σ denoted $A = a_1, \ldots, a_n$ and $B = b_1, \ldots, b_n$. Then, does there exist some sequence of indices (i_k), for $1 \leq k \leq K$ and for $K \geq 1$, and with $1 \leq i_k \leq n$ for all k such that*

$$a_{i_1} \ldots a_{i_K} = b_{i_1} \ldots b_{i_K}?$$

Solution. This problem is shown to be undecidable through a reduction to the halting problem. The proof is technical, and involves encoding the computation history of a Turing machine on an input in such a way that the encoding satisfies the PCP if and only if the Turing machine accepts the input. It is described in detail by Sipser in his book *Introduction to the Theory of Computation* [9]. □

The Post correspondence problem is a useful tool, because it allows us to demonstrate the undecidability of problems without doing the complex reasoning about Turing machines that the halting problem requires. We will now show the undecidability of a variety of problems about context-free grammars through reduction to the Post correspondence problem.

Problem 4.7 (Disjointness). *Let P, Q be CFGs. Is $L(P) \cap L(Q) = \varnothing$?*

Proof of Undecidability. Consider the Post correspondence problem for two lists of words A, B. We construct two context free grammars from these lists as follows:

$$\begin{array}{ll}
G_A \to a_1 1 & G_B \to a_1 1 \\
\quad\vdots & \quad\vdots \\
G_A \to a_n n & G_B \to a_n n \\
G_A \to a_1 G_A 1 & G_B \to a_1 G_B 1 \\
\quad\vdots & \quad\vdots \\
G_A \to a_n G_A n & G_B \to a_n G_B n
\end{array}$$

Then, we can observe that for some string s to exist in both $L(G_A)$ and $L(G_B)$, it must be a solution to the Post correspondence for A, B. So, if $L(G_A) \cap L(G_B) = \varnothing$, then there are no solutions to the Post correspondence for A, B. So, it follows that the Post correspondence problem is reducible to the problem of the disjointness of context-free grammars, so since the Post correspondence problem is undecidable, it follows that the disjointness problem is undecidable. □

Problem 4.8 (Universality). *Let G be some CFG over an alphabet Σ. Then, is*

$$L(G) = \Sigma^*?$$

Proof of Undecidability. Consider the Post correspondence problem for two lists of words A, B. Construct their corresponding grammars G_A, G_B as in Problem 4.7. Then, the languages of these grammars are not just CFLs, but are also Deterministic Context-Free Languages, discussed in detail in Subsection 4.2, since we can construct Deterministic Pushdown Automata to accept them. We have that if

$$\overline{L(G_A) \cap L(G_B)} = \Sigma^*,$$

then

$$L(G_A) \cap L(G_B) = \varnothing.$$

Since these are grammars corresponding to lists in a Post correspondence problem, it follows that

$$\overline{L(G_A) \cap L(G_B)} = \Sigma^*$$

implies that there is no solution to that Post correspondence problem. Additionally, by DeMorgan's Law, we have that

$$\overline{L(G_A) \cap L(G_B)} = \overline{L(G_A)} \cup \overline{L(G_B)}.$$

Since Deterministic Context-Free Languages (DCFLs) are closed under complement and CFLs are closed under union, it follows that $\overline{L(G_A) \cap L(G_B)}$ is a CFL, so the question of whether it is empty is decidable. So, it follows that the problem of universality reduces to Problem 4.6, the Post correspondence problem, which is undecidable. So, we have that the problem of universality is undecidable. □

Definition 4.9. We say that a Context-Free Grammar is *ambiguous* if there is more than one way to apply the production rules to reach an accepted word, assuming a production rule is always applied to the leftmost nonterminal symbol. This is equivalent to stating that there is more than one parse tree that parses an accepted word.

Problem 4.10 (Ambiguity)**.** *Let G be a CFG. Is G ambiguous?*

Proof of Undecidability. Consider the Post correspondence problem for two lists of words A, B. Construct their corresponding grammars G_A, G_B as in Problem 4.7. Now, consider the grammar

$$G \to G_A | G_B.$$

It follows that the ambiguity of G implies that a solution to the Post correspondence problem for A, B exists, so the Post correspondence problem reduces to the ambiguity problem, so the ambiguity problem is undecidable. □

Problem 4.11. *Let G be a CFG. Is $L(G)$ regular?*

Proof of Undecidability. Note that Σ^* is regular, so if $L(G)$ is not regular then it follows that $L(G) \neq \Sigma^*$. Additionally, universality is decidable within regular languages, because they are closed under complement and emptiness is decidable even within context-free grammars. So, we have that Problem 4.8 reduces to the regularity problem. It follows that the regularity problem is undecidable. □

Problem 4.12 (Equality)**.** *Let G_1, G_2 be CFGs. Is $L(G_1) = L(G_2)$?*

Proof of Undecidability. Note that Σ^* is regular, so it is also a context-free. So, we have that Problem 4.8 reduces to the equality problem. It follows that the equality problem is undecidable. □

Problem 4.13 (Inclusion)**.** *Let G_1, G_2 be CFGs. Is $L(G_1) \subseteq L(G_2)$?*

Proof of Undecidability. As before, note that Σ^* is regular, so it is also a context-free. Additionally, observe that for any grammar G, $L(G) \subseteq \Sigma^*$. So, we have that Problem 4.8 reduces to the inclusion problem, since $\Sigma^* \subseteq L(G) \implies \Sigma^* = L(G)$. It follows that the inclusion problem is undecidable. □

4.2. **Problems decidable for DCFLs.**

Definition 4.14. A DCFL is a language accepted by a Deterministic Pushdown Automaton.

DCFLs are another important classification of languages, and are a proper subset of CFLs. They are discussed in detail by Ginsburg [2].

Theorem 4.15 (Closure Properties)**.** *If L is DCFL, then $\overline{L}$ is a DCFL.*

Proof. Let L be a DCFL. Then, there exists some DPDA P that recognizes L. We can construct a new DPDA P' by complementing the accepting states of P. Because P is a DPDA and thus is only in one state at a time, P' will recognize $\overline{L}$ and thus $\overline{L}$ is a DCFL. This would not hold if P were only a PDA, since a nondeterministic PDA accepts if *any* branch ends on an accepting state, and thus complementing the accepting states would not recognize the complement of the originally recognized language. □

Problem 4.16. *Let L be a DCFL. Is $L = \Sigma^*$?*

Proof of Decidability. We have that L is a DCFL, so by Theorem 4.15, $\overline{L}$ is also a DCFL. We know that whether $\overline{L}$ is empty is decidable, so it follows that the question of whether $L = \Sigma^*$ is also decidable. □

Finally, the decidability of the equality problem for DCFLs was an open problem in the field of computability theory from 1965, when it was introduced by Ginsburg and Greibach until 1997, when it was shown to be decidable by Géraud Sénizergues [2, 7]. Sénizergues received the 2002 Gödel Prize for the result.

5. Turing Degrees

In Subsection 3.1, we introduced the concept of Turing reducibility, denoted by the symbol $\leq_T$. This symbol suggests some type of ordering over the set of decision problems. In this section, we introduce that partial ordering on the set of decision problems and examine its properties.

Definition 5.1. Let A and B be decision problems. We say that A and B are *mutually reducibile* and write $=_T$ if $A \leq_T B$ and $B \leq_T A$.

Fact 5.2. The relation $=_T$ is an equivalence relation.

We call the equivalence classes produced by this relation *Turing degrees* or *degrees of unsolvability*, a concept first introduced in 1944 by Emil Post [6]. We write the set of all such degrees as $\mathcal{D}$.

Fact 5.3. The relation $\leq_T$ is a partial ordering of $\mathcal{D}$.

Definition 5.4. An oracle machine for a degree A, denoted $\mathcal{O}_A$, is a Turing machine equipped with a function that is able to decide A.

Definition 5.5. A problem P is called *computably enumerable* or *semidecidable* for some degree $d \in \mathcal{D}$ if there exists a problem $A \in d$ such that it is possible to construct an oracle machine $\mathcal{O}_A$ such that $\mathcal{O}_A$ halts for all the inputs where P is true, and does not halt on all the inputs for which P is false. If $A = \varnothing$, and so the oracle machine does not have an oracle, then we say that P is *Turing recognizable.*

Remark 5.6. The Halting problem, Problem 3.1, is Turing recognizable. The problem is recognized by simulating the input machine.

While it is possible to explicitly state problems lying in the lower degrees, in order to more easily study the general structure of $\mathcal{D}$, we introduce a new operator, *jump*, that increments Turing degrees. Much of the behavior of this operator was shown in a joint paper by Kleene and Post [4].

Definition 5.7. The jump of A, denoted A', is the degree of the halting problem for oracle machines for A.

Fact 5.8. The jump operator has many properties that make it convenient for studying the Turing degrees, some of which are listed below. Further commentary on the jump operator and its properties is available in Robert Soare's book *Turing Computability* [10], as well as in Shore and Slaman's paper *Defining the Turing Jump* [8].

- A' is computably enumerable in A.
- $A' \not\leq_T A$.
- If $B =_T A$ then $B' =_T A'$.

One of the earliest questions about the structure of the Turing degrees was Post's Problem, first posed in 1944 [6]. Post's Problem asks whether there is a computably enumerable degree strictly between 0 and $0'$. A positive answer to the question was found independently by Friedberg and Mucnik in the late 1950s using a technique called the "priority method" [1]. A solution that does not involve a priority argument was provided by Kučera in 1986 [5].

5.1. **Properties and Structure.**

Lemma 5.9. *Every Turing degree is of cardinality* $\aleph_0$.

Proof. We have that since every Turing machine can be expressed as a finite string over $\{0, 1\}$, it can be expressed as a finite subset of $\mathbb{N}$. So, it follows that for any given oracle, there are at most $\aleph_0$ oracle machines, so at most $\aleph_0$ problems are decidable for any Turing degree.

Now that an upper bound has been established, we show that the cardinality of each degree is exactly $\aleph_0$. Suppose, for the sake of contradiction, that a Turing degree contained only a finite number of decision problems. Then, the degree would be decidable, since every finite set is decidable. But we know that the only decidable degree is 0, since all decidable problems are mutually reducible, and in the previous section we showed that the cardinality of 0 is $\aleph_0$, so it follows that every degree has cardinality $\aleph_0$. □

Lemma 5.10. *The set* $\mathcal{D}$ *is of cardinality* $2^{\aleph_0}$.

Proof. First, observe that the cardinality of $\mathcal{D}$ is at most $2^{\aleph_0}$, since it is a set of sets of decision problems and there are at most $2^{\aleph_0}$ decision problems since decision problems are subsets of $\mathbb{N}$.

Now, recall from Lemma 5.9 that each degree contains $\aleph_0$ decision problems. From this, it follows that there must be at least $2^{\aleph_0}$ degrees. So, we have that $|\mathcal{D}| = 2^{\aleph_0}$. □

Definition 5.11. A partially ordered set S is said to form an *upper semilattice* if, for every $x, y \in S$, there exists some element $z \in S$ such that $z \geq x, y$ and for all $w \in A$ such that $w \geq x, y$, we have that $w \geq z$. In other words, $\sup(\{x, y\})$ exists.

Definition 5.12. A partially ordered set S is said to form a *lattice* if all pairs $\{x, y\} \in S$ have both an infinum and a supremum.

Theorem 5.13. *$\mathcal{D}$ forms an upper semilattice.*

Proof. Observe that it is sufficient to prove that every two degrees have a supremum. Let $A, B \in \mathcal{D}$ and $P_1 \in A, P_2 \in B$. Consider the operation

$$A \vee B = \deg\left(\{2n \mid n \in P_1\} \cup \{2n + 1 \cup n \in P_2\}\right),$$

where $\deg(A)$ is the Turing degree of A. Now, we show that $A \vee B = \sup(A, B)$.

First, we show that $A, B \leq_T A \vee B$. This is equivalent to showing that problems in A and B are decidable by oracle machines for $A \vee B$. But simple construction gives oracle machines that can decide P_1, P_2 given an oracle for $A \vee B$: for any n, if $2n \in A \vee B$ then $n \in P_1$, and if $2n + 1 \in A \vee B$ then $n \in P_2$. This holds for any P_1, P_2, so it follows that $A, B \leq_T A \vee B$.

Now, let $W \in \mathcal{D}$ such that $W \geq A, B$ and let $P \in A \vee B$. Then, we have that for every n, we can determine whether $n \in P$ by determining whether n is even and $\frac{n}{2} \in P_1$ or odd and $\frac{n-1}{2} \in P_2$. Since both the parity of a number and P_1 and P_2 are decidable in W, it follows that $A \vee B$ is decidable in W, and thus $A \vee B \leq_T W$. So, it follows that $A \vee B = \sup(A, B)$. This holds for all $A, B \in \mathcal{D}$, so it follows that $\mathcal{D}$ forms an upper semilattice. □

Fact 5.14. $\mathcal{D}$ does not form a lattice.

Remark 5.15. Theorem 5.13 and Fact 5.14 were first shown in Kleene's *The Upper Semi-Lattice of Degrees of Recursive Unsolvability* [4]. A simpler proof of Fact 5.14 is available in Robert Soare's book *Turing Computability* [10, pp. 142–144].

A final result respecting the structure of $\mathcal{D}$ is the existence of *minimal degrees.* A degree A is minimal if $A \neq_T 0$ (A is not Turing-decidable), but there does not exist any degree $B \in \mathcal{D}$ such that $0 <_T B <_T A$. The existence of such degrees was first shown by Spector in 1956 [11].

Acknowledgments

It is a pleasure to thank my mentor, Ronno Das, for supervising this project and providing valuable feedback and advice. I would also like to thank Peter May for organizing this REU, and Daniil Rudenko for running the Apprentice Program.

References

[1] Richard M. Friedberg. "Two recursively enumerable sets of incomparable degrees of unsolvability (solution of Post's problem, 1944)". In: *Proceedings of the National Academy of Sciences* 43.2 (1957), pp. 236–238. ISSN: 0027-8424. DOI: `10.1073/pnas.43.2.236`. eprint: `http://www.pnas.org/content/43/2/236.full.pdf`. URL: `http://www.pnas.org/content/43/2/236`.

[2] Seymour Ginsburg and Sheila A. Greibach. “Deterministic context free languages”. In: *6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)*. Oct. 1965, pp. 203–220. DOI: `10.1109/FOCS.1965.7`.

[3] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 3rd ed. Pearson/Addison Wesley, 2007. ISBN: 0321455363.

[4] S. C. Kleene and Emil L. Post. “The Upper Semi-Lattice of Degrees of Recursive Unsolvability”. In: *The Annals of Mathematics* 59.3 (1954), p. 379. DOI: `10.2307/1969708`. URL: `https://doi.org/10.2307/1969708`.

[5] A Kučera. “An Alternative, Priority-free, Solution to Post’s Problem”. In: *Proceedings of the 12th Symposium on Mathematical Foundations of Computer Science 1986*. Bratislava, Czechoslovakia: Springer-Verlag, 1986, pp. 493–500. ISBN: 0387-16783-8. URL: `http://dl.acm.org/citation.cfm?id=22416.22462`.

[6] Emil L. Post. “Recursively enumerable sets of positive integers and their decision problems”. In: *Bulletin of the American Mathematical Society* 50 (1944), pp. 284–316. ISSN: 1088-9485. DOI: `https://doi.org/10.1090/S0002-9904-1944-08111-1`.

[7] Géraud Sénizergues. “The equivalence problem for deterministic pushdown automata is decidable”. In: *Automata, Languages and Programming*. Ed. by Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 671–681. ISBN: 978-3-540-69194-5.

[8] Richard A. Shore and Theodore A. Slaman. “Defining the turing jump”. In: *Mathematical Research Letters* 6.6 (1999), pp. 711–722. DOI: `http://dx.doi.org/10.4310/MRL.1999.v6.n6.a10`.

[9] Michael Sipser. *Introduction to the Theory of Computation*. 3rd ed. Cengage Learning, 2013. ISBN: 1133187811.

[10] Robert I. Soare. *Turing Computability*. Theory and Applications of Computability. Springer Berlin Heidelberg, 2016. DOI: `10.1007/978-3-642-31933-4`. URL: `https://doi.org/10.1007/978-3-642-31933-4`.

[11] Clifford Spector. “On Degrees of Recursive Unsolvability”. In: *The Annals of Mathematics* 64.3 (1956), p. 581. DOI: `10.2307/1969604`. URL: `https://doi.org/10.2307/1969604`.

[12] A. M. Turing. “On Computable Numbers, With an Application To the Entscheidungsproblem”. In: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265. DOI: `10.1112/plms/s2-42.1.230`. URL: `https://doi.org/10.1112/plms/s2-42.1.230`.