Decidability

William Chan

Preface : In 1928, David Hilbert gave a challenge known as the Entscheidungsproblem, which is German for Decision Problem. Hilbert's problem asked for some purely mechanical procedure (devoid of all ingenuity) such that when given the description of a formal language (usually first-order logic) and some well-formed formula in that language, it would determine if that statement can be proven (is deducible). Briefly, a formal system includes a set of symbols, a set of finite sequences of those symbols known as well-formed formulas (usually a way to determine mechanically whether a finite sequence is well-formed), a set of well-formed formulas known as axioms, and a finite set of relations called rules of inference (modus ponen is an example). A proof of a well-formed formula is a finite sequence of statements such that each statement is an axiom or a consequence of previous statements according to a rule of inference. Therefore, the Entscheidungsproblem asks for a procedure that will determine whether a well-formed formula in a formal system has a proof. This mechanical procedure is commonly known as an algorithm.

In order to attempt this challenge, logicians had to formalize the intuitive idea of an algorithm. Several different approaches were developed. Alonzo Church created the λ-calculus. Kleene (based on works by Kurt Gödel) created the general recursive function. Alonzo Church, using his λ-calculus, gave a negative answer to the Entscheidungsproblem for the formula system of number theory. However, many logicians, prominently Gödel, were not convinced that λ-calculus or the general recursive function described everything that could be done algorithmically or computed. Alan Turing soon created what is known as the Turing Machine. Of the previous models of computation, Turing's was the most intuitive. The Turing Machine, more so than the others, captured the theoretical concept of a being following directions and computing. The Turing Machine imitated human computation or the intuitive idea of an algorithm, in that there existed a finite set of directions. It idealized the notion of computation by allowing the device to run for an indefinite number of steps and had an infinite amount of space or memory to store symbols. Because of its intuitive nature, Gödel was convinced that this captured the intuitive idea of an algorithm. Furthermore, it has been proved that the Turing Machine, λ-Calculus, General Recursive Function, and many other models are equivalent in that they can, informally, imitate each other. The Church-Turing Thesis, which is more of a philosophical statement than a mathematical statement, asserts this claim.

This paper will not consider the philosophical, intuitive notion of an algorithm. This paper shall define an algorithm to be something that can be performed using the formal model of a Turing Machine. The Church-Turing Thesis will never be explicitly used; however, the spirit of its assertion will frequently be applied - mostly due to the author's laziness. One can see the paper's construction of a Universal Turing Machine or any of the numerous examples to observe that simple procedures (such as adding) may have very complicated and lengthy Turing Machine programs.

This paper will prove basic results and limitations of algorithms. The first section introduces the basic concepts and terminologies used throughout the paper. Note $\mathbb{N} = \{0, 1, 2, ...\}$ will denote the set of natural numbers. Section 2 introduces a weak (but simple) model of computation, known as the Finite Automaton, which will yield many examples in the study of algorithms. Section 3 gives the definition of a Turing Machine (and other equivalent models) and various associated notions. Section 4 illustrates the capabilities and limitations of algorithms. Finally, Section 5 discusses computable functions.

Table of Contents

Section 1: Basics

Remarks : A set is a collection of objects known as elements.

Definition 1.1 : If A is a set and a is an element of A, then one denotes this $a \in A$. $\notin$ will mean not an element of. If A and B are two sets, and every element of B is an element of A, then one says that B is a subset of A, denoted $B \subset A$. If A and B are sets and A and B have the same elements, then one says that A and B are identical, and one represents this symbolically as $A = B$. Similarly, $\neq$ will represent non-set equality. The set with no elements is $\emptyset$. The power set of A, denoted $\mathscr{P}(A)$ is the set of all subsets of A. Note $\emptyset \subset A$.

Theorem 1.2 : Let A and B be sets, $A = B$ if and only if $A \subset B$ and $B \subset A$.

Proof : Suppose $A = B$. By definition, A and B have the exact same elements. Therefore, $A \subset B$ since every element of A is an element of B. Similarly for $B \subset A$.

Suppose $A \subset B$ and $B \subset A$. Suppose that $A \neq B$, then without loss of generality, suppose that there exists $a \in A$ such that $a \notin B$. Immediately, one sees that every element of A (in particular a) is not an element of B. This means one has not $A \subset B$, contradicting the hypothesis. ■

Definition 1.3 : Suppose that A and B are sets. $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$ and is called the union of the two sets. The intersection is $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$. The set difference $A \backslash B = \{x \mid x \in A \text{ and } x \notin B\}$. Let X be a set sometimes called the universe. Let $A \subset X$, one defines the complement of A (in X) as $A^{c(X)} = X \backslash A$. When the universe is clear from context, this paper shall simply write A^c.

Remarks : "or" in this paper will always be inclusive. That is, a or b holds when a holds, b holds, or both.

Definition 1.4 : Let A be a set and let $a \in A$ and $b \in A$. An ordered pair with a first and b second, denoted (a, b), is the set $\{\{a\}, \{a, b\}\}$. Similarly, one can define n-tuples for $n \in \mathbb{N}$.

Theorem 1.5 : Let A be a set and $a, b, c, d \in A$. $(a, b) = (c, d)$ if and only if $a = c$ and $b = d$.

Proof : Suppose $(a, b) = (c, d)$. By definition $\{\{a\}, \{a, b\}\} = \{\{c\}, \{c, d\}\}$. If these two sets are the same, they must have the same elements. One has that $\{a\} = \{c\}$, because $\{a\}$ can not equal $\{c, d\}$ since one is a singleton and the other is a two element set. Thus $a = c$. Immediately, one has that $\{a, b\} = \{c, d\}$. Suppose $b \neq d$, then because these sets are equal, b must equal something from the other set. One is left with $b = c$. However, one has already shown that $c = a$, so by the transitivity of equality, $b = a$. So for the set equality to hold, one must have that $c = d$. We have shown $b = a$, $b = c$, and $b = d$, so by transitivity $b = d$, contradicting our assumption. ■

Definition 1.6 : Let A and B be sets. The cartesian product of A and B denoted $A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$. If $A_1, A_2, ..., A_n$, for $n \in \mathbb{N}$, are sets, one could similarly define $A_1 \times ... \times A_n$. If $A = B$, one writes $A^2 = A \times A$. Similarly, A^n.

Definition 1.7 : For A a set and $n \in \mathbb{N}$, a n-place relation R on A is a subset of A^n. Let A and B be sets and $B \neq \emptyset$, a function f from A to B, denoted $f : A \to B$ is a 2-place relation such that if $(a, b_1), (a, b_2) \in f$,

then $b_1 = b_2$. From this point on, one will actually think of functions in the usual way. A is called the domain. B is called the codomain. For $C \subset A$, $f(C) = \{b \mid b = f(c) \text{ for some } c \in C\}$. $f(A)$ is the range of f. A function f is surjective if for all $b \in B$ there exists $a \in A$ such that $f(a) = b$ - or rather $f(A) = B$. A function f is injective if for $a, a' \in A$ and $a \neq a'$, then $f(a) \neq f(a')$. A function f is bijective, if it is both surjective and injective.

Definition 1.8 : Let A and B be sets, if there exists a bijection from A to B, one says that A and B have the same cardinality. The cardinality of A is denoted $|A|$. If A is a finite set with n elements, one says that $|A| = n$. If A has the same cardinality as $\mathbb{N}$, one says that $|A| = \aleph_0$. Sets with cardinality $\aleph_0$ are also said to be countably infinite. Countable sets are sets that are finite or countably infinite.

Theorem 1.9 : Let $A_1, A_2, ...$ be a sequence of countable sets, then $\bigcup_{n\in\mathbb{N}} A_i$ is countable.

Proof : If A_i is countable, then one can enumerate all the elements of A_i in a list: $a_{i,1}, a_{i,2}, ...$ Then display all the elements of the union in the following array:

$$\begin{array}{cccccc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & \cdots \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & \cdots \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & \cdots \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & \cdots \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \end{array}$$

One would enumerate the union by going along the diagonals. That is, the list would be $a_{1,1}, a_{2,1}, a_{1,2}, a_{3,1}, a_{2,2}, a_{1,3}, ...$ The bijection follows from this list. ■

Theorem 1.10 : Let $A_1, A_2, ..., A_n$ be a finite number of countable sets. Then $A_1 \times A_2 \times ... \times A_n$ is countable.

Proof : For $k = 1$, this is obvious since A_1 is already countable. Suppose this hold for $A_1 \times ... \times A_{n-1}$. Note that

$$A_1 \times ... \times A_n = \bigcup_{a\in A_n} (A_1 \times ...A_{n-1}) \times \{a\}$$

By the induction hypothesis, $A_1 \times ... \times A_{n-1}$ is countable. Cartesian product this set with the singleton is still countable. Since A_n is countable, one is merely considering the countable union of countable sets; therefore, by Theorem 1.9, this set is countable. The theorem follows from induction. ■

Theorem 1.11 : Let A and B be two finite sets. The set of all functions $f : A \to B$ is a finite set of cardinality $|B|^{|A|}$.

Proof : f is determined by how it maps each element in the domain. For each $a \in A$, $f(a)$ can be any element of B. Therefore, there are $|B|$ number of possibilities for $f(a)$. Then there are $|A|$ number of elements in the domain which each has $|B|$ number of element of B to which it can be mapped. Hence the total number of functions is $|B|^{|A|}$. ■

Theorem 1.12 : The set of all infinite sequences consisting of only two distinct elements is uncountable

(not countable).

Proof : Other than how one names the elements, all sets of two elements are the same. Let the two distinct element be 0 and 1. Suppose the set of all infinite sequences made of these two elements are countable. This implies that they can be placed into bijective correspondence with $\mathbb{N}$. That is to say, they can be listed. Suppose one has listed all such infinite sequences in the following array

$$\begin{array}{cccccc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & \cdots \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & \cdots \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & \cdots \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & \cdots \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \end{array}$$

Now construct a new sequence $b_1, b_2, ...$ such that if $a_{i,i} = 1$, let $b_i = 0$, and if $a_{i,i} = 0$, let $b_i = 1$. This is an infinite sequence consisting of 0 and 1. Where does this sequence occur on the list? It can not be the first sequence, since it differs from it by at least the first term. It can not be the i^{th} sequence because it differs from it by at least the i^{th} term. Therefore this sequence does not occur on our list. However, the assumption was that this list enumerated every possible sequence. Contradiction. ■

Remark : One can also modify this proof to show that $\mathbb{R}$ is uncountable. We could represent each real number by one of its decimal representation. Suppose it did lie in some list, then construct a new decimal, real number by choosing an element from $\{1, 2, 3, 4, 5, 6, 7, 8\}$ that is not the same as the entry in the diagonal of the array. One excludes 0 and 9, because real numbers do not have unique decimal representation. For instance, $2.99999... = 3.0000...$

Theorem 1.13 : Let A be a set. A and $\mathscr{P}(A)$ do not have the same cardinality.

Proof : Suppose that A and $\mathscr{P}(A)$ have the same cardinality. Therefore, there exists a bijecion $\phi : A \to \mathscr{P}(A)$. Note that for each a, $\phi(a) \in \mathscr{P}(A)$ which means that $\phi(A)$ is a subset of A. Now define $F = \{a \mid a \in A \text{ and } a \notin \phi(A)\}$. Certainly $F \subset A$; therefore, $F \in \mathscr{P}(A)$. Because ϕ is a bijection (specifically surjective), there exists $f \in A$ such that $\phi(f) = F$. Now consider whether $f \in F$. There are two possiblities. $f \in F = \phi(f)$. If this is true then $f \notin F = \phi(f)$ since by definition F is the set of all elements of A which are not in its own image. Suppose $f \notin F$. Then $f \in F$ because by definition of F, it contains all elements of A which are not in its own image. Contradiction since f is either in a set or not in that set. ■

Definition 1.14 : An alphabet is a nonempty, finite set. The elements of an alphabet are called the symbols. A string over an alphabet is a finite sequence of symbols from that alphabet. The length of a string is the number of symbols occurring in that string. The empty string denoted ϵ is the string of length zero. A language is a set of strings.

Definition 1.15 : A partial ordering of a set A is a 2-place relation $\leq$ on A with the following property: (1) For all $a \in A$, $a \leq a$. Reflexive (2) For all $a, b \in A$, if $a \leq b$ and $b \leq a$, then $a = b$. Antisymmetry (3) For all $a, b, c \in A$, if $a \leq b$ and $b \leq c$, then $a \leq c$. Transitivity. A total ordering of A is a partial ordering such that every two elements of A are comparable, that is if $a, b \in A$, then $a \leq b$ or $b \leq a$.

Definition 1.16 : Suppose an alphabet Λ has a total ordering. The lexicograpical ordering of the strings over Λ is an ordering such that strings of shorter lengths precede longer ones, and two strings $a_1a_2...a_n \leq b_1b_2...b_n$ if and only if there is some i such that $1 \leq i \leq n$ such that $a_i < b_i$ and for all $j \geq 1$, $j < i$, $a_j = b_j$.

Section 2: Finite Automata

Definition 2.1 : A Finite Automaton is a 5-tuple $(\Sigma, \Lambda, \delta, s_0, F)$, where Σ is a finite set of elements called states, Λ is a finite set (called the alphabet), $\delta : \Sigma \times \Lambda \to \Sigma$ is the transition function, $s_0 \in \Sigma$ is the start state, and $F \subset \Sigma$ is the set of final (or accept) states.

Remarks : To interprete this definition, a finite automaton begins in the start state s_0. It takes in some string $w = w_1 w_2 ... w_n$ over the alphabet Λ. It consults its transition function to see where it goes next. That is, at the beginning, the finite automaton is at state s_0 and is reading the first symbol w_1 of the input string w. If $\delta(s_0, w_1) = s$, then the automaton goes into state s. Its next move would be determined by $\delta(s, w_2)$. At the end of the input string, if the finite automaton is in a final state, then one says that the finite automaton accepts the string w.

Definition 2.2 : Let $B = \{\Sigma, \Lambda, \delta, s_0, F\}$ be a finite automaton. B is said to accept input $w = w_1 w_2 ... w_n$ if there is a sequence of states $r_0, ..., r_n$ such that $r_0 = s_0$, the accept state, $\delta(r_i, w_{i+1}) = r_{i+1}$, and $r_n \in F$.

Definition 2.3 : Let A be a finite automaton. One says that A recognizes the language L if $L = \{w \mid w \text{ is a string that } L \text{ accepts.}\}$. A language is called a regular language if some finite automaton recognizes it. The language that the finite automaton A recognizes is denoted $L(A)$

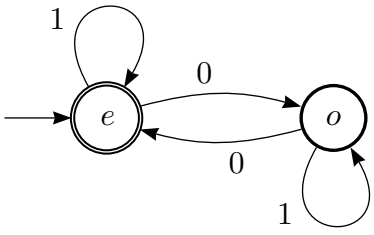
Remarks : A illustrative way of viewing a finite automaton is using a state diagram. In this diagram, the states are symbolized by circles over the respective state name. The start state is indicated by the circle with an arrow pointing to it from no other state. The final states are indicated by the double circled symbols. The transition function is represented by the arrows between states. Next to arrows, there are alphabet symbols. Suppose on state s_0, one sees an arrow labeled a pointing to state s_3. This means that when the finite automaton is on state s_0 and reading the symbol a on the input string, the automaton moves to state s_3. The finite automaton accepts a particular string, if at the end of reading the input string, the automaton is on a final state.

Theorem 2.4 : Let L be the following language over $\Lambda = \{0, 1\}$:

$$L = \{w \mid w \text{ contains an even number of } 0\}$$

L is a regular language.

Proof : The finite automaton B that recognizes L is represented by the following state diagram.



The start state is e. The first time that B reads 0, it moves to state o indicating it has read an odd number of 0 thus far. The next time it read a 0, it goes back to e because it has seen an even number of 0. On state o or e, reading a 1 merely causes B to remain on the current state, since the number of 0 read has not changed. The set of the final states is the singleton $\{e\}$.

The formal description of B is $B = \{\{e, o\}, \{0, 1\}, \delta, e, \{e\}\}$, where $\delta = \{(e, 0, o), (e, 1, e), (o, 0, e), (o, 1, o)\}$.

■

Remarks : In general, automata are abstract computation machines or models. Two automata are said to be equivalent if they recognize the same language.

A finite automaton is said to be deterministic if given the current state of the machine and the next input symbol, there is exactly one state to which to change. A finite automaton is nondeterministic if there are no or several choices of next states given a current state and input symbol. The finite automaton seen above is a deterministic finite automaton. All deterministic finite automaton are nondeterministic finite automaton.

The key differences between the two is that nondeterministic finite automata have zero or many arrows leaving a given state for each symbol of the alphabet (on its state diagram). Moreover, the nondeterministic finite automaton may have arrows labeled with the empty string ϵ. Again, from any given state, there may be zero or many arrows labeled with ϵ.

Informally, when a nondeterministic finite automaton reaches a state and read an input symbol with multiple ways of proceding, the automaton splits into multiple branches of computation and continues seperately along each. If the automaton happens to be on a state with an arrow labeled ϵ leaving it, then before reading the next symbol, it splits into another branch going along that ϵ arrow. If an automaton has no arrows corresponding to its current state and input symbol, then that particular branch of the computation stops. The nondeterministic finite automaton is said to accept a particular input, if at least one of the computation branches accept. Otherwise, the nondeterministic finite automaton rejects.

Definition 2.5 : A Nondeterministic Finite Automaton is a 5-tuple $(\Sigma, \Lambda, \delta, s_0, F)$ where Σ is a finite set of elements, known as states, Λ is a finite set (called the alphabet) of elements called symbols, $\delta : \Sigma \times \Lambda' \to \mathscr{P}(\Sigma)$, where $\Lambda' = \{\Lambda\} \cup \{\epsilon\}$, $s_0 \in \Sigma$ is the start state, and $F \subset \Sigma$ is the set of final (or accepting) states.

Remarks : Note that the codomain of the transition function is now the power set of Σ. So the image is a subset of Σ. This represent all possible next states given the current state and the input symbol.

Definition 2.6 : Let $N = \{\Sigma, \Lambda, \delta, s_0, F\}$ be a nondeterminsitic finite automaton. N is said to accept input $w = w_1w_2w_3...w_n$ if one can write $w = y_1y_2...y_m$ where $y_i \in \Lambda'$ (that is adding ϵ symbol between any $w_i \in \Lambda$) and a sequence of states $r_0, r_1, ..., r_m$ such that $r_0 = s_0$, $r_{i+1} \in \delta(r_i, y_{i+1})$ (for $0 \leq i \leq m-1$), and $r_m \in F$.

Theorem 2.7 : A nondeterministic finite automaton is equivalent to a deterministic finite automaton.

Proof : It is clear that all deterministic finite automaton are nondeterministic.

Let $N = \{\Sigma, \Lambda, \delta, s_0, F\}$ be a nondeterministic finite automaton that recognizes some language $L(N)$. One seeks to create a deterministic finite automaton $D = \{\Sigma', \Lambda', \delta', s_0', F'\}$ which recognizes $L(N)$.

To construct D, one lets $\Sigma' = \mathscr{P}(\Sigma)$. The alphabet $\Lambda' = \Lambda$. Define for all $S \in \Sigma'$, $E(S) = \{s \mid s \in \Sigma$ and s can be reached from a state in S by traveling along zero or more ϵ arrows$\}$. The transition function is $\delta : \Sigma' \times \Lambda' \to \Sigma'$ defined by

$$\delta'(S, \omega) = \bigcup_{s \in S} E(\delta(s, \omega))$$

for $S \in \Sigma'$ and $\omega \in \Lambda'$. That is, the transition function is the union of all the sets $E(\delta(s, \omega))$ for each state $s \in S$. The start state $s_0' = E(\{s_0\})$. Finally, the set of final states $F' = \{S \in \Sigma' \mid S$ has an accept state of $N\}$.

D clearly accepts the same language as N. D starts with the set containing the start state of N and all other states reachable from the start state by traveling only along ϵ arrows. Then the transition funtion of D keeps track of all the possible branches of N by taking the union of $E(\delta(s, \omega))$ for all $s \in S$, the current state of D and $\omega \in \Lambda$. The transition function accounts also for any states reachable by ϵ arrows. The accepting state contains any subset of Σ which has an accept state of N. So if any branch of N on a

particular computation accepts, the state of D, which is really a set of states of N, would contain at least one accepting state. Thus whatever, N accepts, D does as well. ■

Theorem 2.8 : A language is regular if and only if some nondeterministic finite automaton recognizes it.

Proof : If a language is regular, it is recognized by a deterministic finite automaton by definition. Every deterministic finite automaton is a nondeterministic finite automaton.

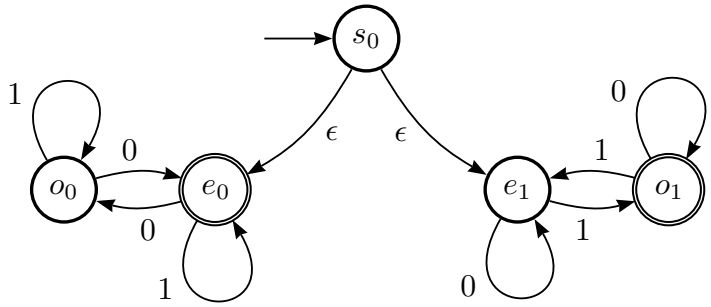
By Theorem 2.7, every nondeterministic finite automaton has an equivalent deterministic finite automaton. If the nondeterministic finite automaton recognizes some language, then its equivalent deterministic finite automaton recognizes that language; hence, the language is regular by definition. ■

Theorem 2.9 : Let $\Lambda = \{0, 1\}$. Let L' be the language over Λ defined by

$$L' = \{w \mid w \text{ contains an even number of 0 or an odd number of 1}\}$$

L' is a regular language.

Proof : The state diagram of the nondeterministic finite automaton is as follows:

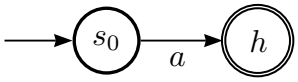


The start state is s_0. Before any computation, the two ϵ arrows indicate a break to two branches. The left side keeps track of whether the automaton has read an even or an odd number of zeros. The right side keeps track of whether the machine has read an even or odd number of ones. Note that the left side is exactly the same as the diagram in Theorem 2.4 and the right is the analagous diagram for determining odd number of ones. The set of accept states are e_0 and o_1, representing even number of 0 and odd number of 1, respectively.

The formal description of this nondeterministic finite automaton is
$\{\{s_0, e_0, e_1, o_0, o_1\}, \{0, 1\}, \delta, s_0, \{e_0, o_0\}\}$, where $\delta = \{\{s_0, \epsilon, \{e_0, e_1\}\}, \{s_0, 0, \emptyset\}, \{s_0, 1, \emptyset\}, \{e_0, \epsilon, \emptyset\}, \{e_0, 0, \{o_0\}\}, \{e_0, 1, \{e_0\}\}, \{e_1, \epsilon, \emptyset\}, \{e_1, 0, \{e_1\}\}, \{e_1, 1, \{o_1\}\}, \{o_0, \epsilon, \emptyset\}, \{o_0, 0, \{e_0\}\}, \{o_0, 1, \{o_0\}\}, \{o_1, \epsilon, \emptyset\}, \{o_1, 0, \{o_1\}\}, \{o_1, 1, \{e_1\}\}\}$. ■

Theorem 2.10 : Let Λ be an nonempty alphabet. Let $a \in \Lambda$, then $\{a\}$ is a regular language.

Proof : Consider the nondeterministic finite automaton :



The formal description of this finite automaton is $\{\{s_0, h\}, \Lambda, \delta, s_0, \{h\}\}$, where

$$\delta(x) = \begin{cases} \{h\} & \text{if } x = (s_0, a) \\ \emptyset & \text{if } x = (s_0, \beta) \text{ where } \beta \in \Lambda \text{ and } \beta \neq a \\ \emptyset & \text{if } x = (h, \gamma) \text{ for all } \gamma \in \Lambda \end{cases}$$

■

Definition 2.11 : Let A and B be languages. The union operation is a binary (taking two arguments) operation on the set of languages, denoted by $A \cup B$, defined as follows: $A \cup B = \{w \mid w \in A \text{ or } w \in B\}$.

Remarks : Suppose $A = \{\star, \circ\}$ and $B = \{\text{cat}, \text{dog}\}$, then $A \cup B = \{\star, \circ, \text{cat}, \text{dog}\}$.

Definition 2.12 : Let A and B be languages. The catenation operation is a binary operation on the set of languages, denoted by $A \circ B$ (sometimes written just AB), defined as follows: $A \circ B = \{vw \mid v \in A \text{ and } w \in B\}$.

Remarks : Suppose that $A = \{\text{good}, \text{bad}\}$ and $B = \{\text{cat}, \text{dog}\}$ are languages over the alphabet $\Lambda = \{a, b, c, ...z\}$, then $A \circ B = \{\text{goodcat}, \text{gooddog}, \text{badcat}, \text{baddog}\}$.

Definition 2.13 : Let A be a language. The Kleene Star (or just Star) operation is a unary (taking one argument) operation on the set of languages, denoted by *, defined as follows: $A^* = \{w_1 w_2 ... w_k \mid k \geq 0 \text{ and each } w_i \in A\}$.

Remarks : Note that for any set A, $\epsilon \in A^*$. Suppose that $A = \{\text{stupid}, \text{philosophy}, \text{is}\}$ is a language over $\Lambda = \{a, b, c, ..., z\}$, then
$A^* = \{\epsilon, \text{philosophy}, \text{is}, \text{stupid}, \text{philosophyphilsophy}, \text{philosophyis}, \text{philosophystupid}, \text{isis}, \text{isphilosophy}, \text{isstupid}, \text{stupidstupid}, \text{stupidphilosophy}, \text{stupidis}, \text{philosophyphilsophyphilsophy}, \text{philosophyphilosphyis}, \text{philosophyphilosophystupid}, \text{philosophyisphilosophy}, \text{philosophyisis}, \text{philosophyisstupid}, ...\}$.

Theorem 2.14 : Regular languages are closed under the union operation.

Proof : Let A and B be regular languages. This means that A and B are recognized by deterministic finite automata $D_1 = \{\Sigma_1, \Lambda_1, \delta_1, s_1, F_1\}$ and $D_2 = \{\Sigma_2, \Lambda_2, \delta_2, s_2, F_2\}$, respectively. The nondeterministic finite automaton $N = \{\Sigma, \Lambda, \delta, s_0, F\}$ which recognizes the union language will have a new start state, called s_0, which branches out by ϵ arrows to the two old start states. The nondeterministic finite automaton runs D_1 and D_2 on parallel branches. Clearly, N will accept an input w if and only if w is accept by D_1 or D_2.

Formally, $\Sigma = \{s_0\} \cup \Sigma_1 \cup \Sigma_2$. $\Lambda = \Lambda_1 \cup \Lambda_2$. The start state of N is the new start state s_0. The set of

accept states is $F = F_1 \cup F_2$. The transition function δ is

$$\delta(s,a) = \begin{cases} \delta_1(s,a) & \text{if } s \in \Sigma_1 \text{ and } a \in \Lambda_1 \\ \delta_2(s,a) & \text{if } s \in \Sigma_1 \text{ and } a \in \Lambda_2 \\ \{s_1, s_2\} & \text{if } s = s_0 \text{ and } a = \epsilon \\ \emptyset & \text{if } s = s_0 \text{ and } a \neq \epsilon \\ \emptyset & \text{if } s \in \Sigma_1 \text{ and } a \in \Lambda_2 \backslash \Lambda_1 \\ \emptyset & \text{if } s \in \Sigma_2 \text{ and } a \in \Lambda_1 \backslash \lambda_2 \\ \emptyset & \text{if } s \in \Sigma_1 \cup \Sigma_1 \text{ and } a = \epsilon \end{cases}$$

■

Theorem 2.15 : Regular languages are closed under the concatenation operation.

Proof : Suppose that A and B are regular languages, recognized by two deterministic finite automaton $D_1 = \{\Sigma_1, \Lambda_1, \delta_1, s_1, F_1\}$ and $D_2 = \{\Sigma_2, \Lambda_2, \delta_2, s_2, F_2\}$, respectively. The nondeterministic finite automaton which recognizes the languages $A \circ B$ has the same start state as D_1. On an input, it runs D_1 first. Then from all the accepts states of D_1 there are ϵ arrows going to the start state of D_2. Then N runs the input (or the remainder of that input on a seperate branch) on D_2. The construction of N is as follows:

$\Sigma = \Sigma_1 \cup \Sigma_2$. $\Lambda = \Lambda_1 \cup \Lambda_2$. The start state $s_0 = s_1$ of D_1. The accept state $F = F_2$ of D_2. The transition function δ is defined as follows:

$$\delta(s,a) = \begin{cases} \delta_1(s,a) & \text{if } s \in \Sigma_1,\ s \notin F_1, \text{ and } a \in \Lambda_1 \\ \delta_1(s,a) & \text{if } s \in F_1, \text{ and } a \in \Lambda_1 \\ \delta_1(s,a) \cup \{s_2\} & \text{if } s \in F_1, \text{ and } a = \epsilon \\ \delta_2(s,a) & \text{if } s \in \Sigma_2,\ a \in \Lambda_2 \\ \emptyset & \text{if } s \in \Sigma_1 \text{ and } a \in \Lambda_2 \backslash \Lambda_1 \\ \emptyset & \text{if } s \in \Sigma_2 \text{ and } a \in \Lambda_1 \backslash \Lambda_2 \\ \emptyset & \text{if } s \in (\Sigma_1 \cup \Sigma_2) \backslash (F_1) \text{ and } a = \epsilon \end{cases}$$

■

Theorem 2.16 : Regular languages are closed under the star operation.

Proof : Suppose A is a regular language recognized by the deterministic finite automaton $D = \{\Sigma', \Lambda', \delta', s_0', F'\}$. One wishes to construct a nondeterministic finite automaton $N = \{\Sigma, \Lambda, \delta, s_0, F\}$ which will recognize A^*. First, note that ϵ is in A^*. To account for the empty string, one adds a new state s_0 which will also be a final state. There is an ϵ arrow from this new start state to the old start state of D. Then one lets D run the input string; however, one make ϵ arrows from each of the accept state of D back to the original start state. This allows N to accept strings that are multiple strings of A following each other.

The formal construction of N is as follows. $\Sigma = \{s_0\} \cup \Sigma'$. $\Lambda = \Lambda'$. The new state s_0 is the start state of N. The set of final states are $F = \{s_0\} \cup F'$. The transition function δ is defined as follows:

$$\delta(s,a) = \begin{cases} \delta'(s,a) & \text{if } s \in \Sigma' \text{ and } a \neq \epsilon \\ \delta'(s,a) \cup \{s_0'\} & \text{if } s \in F' \text{ and } a = \epsilon \\ \{s_0'\} & \text{if } s = s_0 \text{ and } a = \epsilon \\ \emptyset & \text{if } s = s_0 \text{ and } a \neq \epsilon \\ \emptyset & \text{if } s \in \Sigma' \backslash F' \text{ and } a = \epsilon \end{cases}$$

■

Theorem 2.17 : Let Λ be some nonempty alphabet. Then $\Lambda^* = \{w \mid w \text{ is a string over the alphabet } \Lambda\}$ is

a regular language.

Proof : By Theorem 2.10, the set $\{a\}$, where $a \in \Lambda$ is a regular language. The set $\{a \mid a \in \Lambda\} = \Lambda$ is regular language because it is equal to

$$\bigcup_{a \in \Lambda} \{a\}$$

which is a union of regular language and regular by Theorem 2.14. Λ^* is then regular because it is the star of a regular set. ■

Definition 2.18 : Given a language A over a alphabet Λ, the complement of A, denoted A^c, is $A^{c(\Lambda^*)}$, that is the complement of A in Λ^*.

Theorem 2.19 : Regular languages are closed under complementation.

Proof : Let A be a regular langauge recognized by $D = \{\Sigma, \Lambda, \delta, s_0, F\}$ deterministic finite automaton. The deterministic finite automaton B that recognizes A^c is the automaton that has as its accept states all the non-accept states of D.

Formally $B = \{\Sigma, \Lambda, \delta, s_0, \Sigma \backslash F\}$. ■

Theorem 2.20 : If A, B, and X are sets such that $A \subset X$ and $B \subset X$, then $A \cap B = (A^c \cup B^c)^c$.

Proof : If $x \in A \cap B$, then $x \in A$ and $x \in B$. Therefore, $x \notin A^c$ and $x \notin B^c$. This means $x \notin A^c \cup B^c$. Therefore, $x \in (A^c \cup B^c)^c$. This proves $A \cap B \subset (A^c \cup B^c)^c$.

If $x \in (A^c \cup B^c)^c$, then $x \notin A^c \cup B^c$. So $x \notin A^c$ and $x \notin B^c$. So $x \in A$ and $x \in B$, which means $x \in A \cap B$. $(A^c \cup B^c)^c \subset A \cap B$. ■

Theorem 2.21 : Regular languages are closed under intersection.

Proof : Suppose that A and B and regular language. Because regular langauges are closed under union and complementation, by Theorem 2.20, regular languages are closed under intersection. ■

Definition 2.22 : Let A and B be sets, the symmmetric difference of A and B, denoted $A \Delta B$, is $(A \cap B^c) \cup (A^c \cap B)$.

Theorem 2.23 : Regular languages are closed under symmetric differences (where complement of languages is defined above).

Proof : Regular languages are closed under complementation so B^c and A^c are regular. Regular languages are closed under intersection by Theorem 2.21. Therefore $(A \cap B^c)$ and $(A^c \cap B)$ are regular. Because regular languages are closed under union, $(A \cap B^c) \cup (A^c \cap B)$ is regular. ■

Theorem 2.24 : (Pumping Lemma) If A is a regular language, then there is a number p (called the pumping length) such that if $w \in A$ and w has length greater than or equal to p, then w can be divided into three

substrings (x, y, and z) in the form $w = xyz$, satisfying the following condition
(1) For $i \geq 0$, $xy^iz \in A$
(2) $|y| > 0$
(3) $|xy| \leq p$

Proof : Let $D = \{\Sigma, \Lambda, \delta, s_0, F\}$ be a deterministic finite automaton that recognizes A. Let p be $|\Sigma|$, that is the number of state of D. Let $w = w_1w_2...w_n$ be a string of A with length $n \geq p$. Let $r_1, ..., r_{n+1}$ be the accepting sequence of states such that $r_1 = s_0$, $r_{i+1} = \delta(r_i, w_i)$ for $0 \leq i \leq n$, and $r_{n+1} \in F$. Because this sequence of states has length $n + 1 \geq p + 1$ and D has only p states, one or more of the states must repeat (by the pigeonhole principle). Let the first repeated state occur at r_a and r_b, $0 \leq a < b \leq n + 1$. In fact, the repeat must occur before the $(p + 1)^{\text{th}}$ state, so $b \leq p + 1$. Now define $x = w_1...w_{a-1}$, $y = w_a...w_{b-1}$, and $z = w_b...w_n$.

To see that $xy^iz \in A$, note that the substring x takes B from state r_1 to state r_a. y takes B from state r_a to state r_b. Since $r_a = r_b$, one can say that y takes B from state r_a back to r_a. y^i for $i \geq 0$ will just take the B back to r_a many times. Finally z takes B from $r_a = r_b$ to r_{n+1} an accept state. Therefore, $xy^iz \in A$. Condition (2) is satisfied because $a \neq b$ so there is a string of at least one symbol that takes B from state r_a to r_b. Finally, note that $b \leq p + 1$ and $xy = w_1...w_{b-1}$, so $|xy| \leq p$. ■

Theorem 2.25 : Let $\Lambda = \{0\}$ be the single symbol alphabet. Let $K = \{0^{n^2} \mid n \geq 0\}$. K is not a regular language.

Proof : K is the set of all strings made by a perfect square number of zeros. Now for any p, consider $0^{p^2} \in K$, consider all division of $0^{p^2} = xyz$ such that $|xy| \leq p$ and $|y| > 0$. This implies that $0 < |y| \leq p$. $|xy^2z| \leq p^2 + p < p^2 + 2p + 1 = (p + 1)^2$. However, since $|y| > 0$, $p^2 = |xyz| < |xy^2z| < (p + 1)^2$. This shows that $|xy^2z|$ is strictly between two perfect squares, i.e. it is not a perfect square. Therefore, $xy^2z \notin K$. By the contrapositive of the Pumping Lemma, K can not be a regular language. ■

Theorem 2.26 : Let $\Lambda = \{0, 1\}$ be an alphabet. The langauge $J = \{w \mid w \text{ has an equal number of 0 and 1}\}$ is not a regular langauge.

Proof : Let p be any number. Consider the string 0^p1^p. Consider all division of $0^p1^p = xyz$ such that $|y| > 0$ and $|xy| \leq p$. Because $|xy| \leq p$, one knows that y is a substring consisting of only zeros. Because $|y| > 0$, it contains at least one 0. Consider xy^2z. It has at least one more zero than xyz so xy^2z has unequal number of 0 and 1. $xy^2z \notin J$; therefore, by the contrapositive of the Pumping Lemma, it is not a regular language. ■

Section 3: Turing Machines

Definition 3.1 : A Turing Machine is a 7-tuple $(\Sigma, \Lambda, \Gamma, \delta, s_0, s_\star, s_\times)$ where Σ is a finite set of objects called states, Λ is a finite set (called the Alphabet) of objects such that $\sqcup \notin \Lambda$ (the blank symbol), Γ is a finite set (called the Tape Alphabet) such that $\sqcup \in \Gamma$ and $\Lambda \subset \Gamma$, $\delta : \Sigma \times \Gamma \to \Sigma \times \Gamma \times \{\text{L,R}\}$ called the transition function, $s_0 \in \Sigma$ is the start state, $s_\star \in \Sigma$ is the accept state, and $s_\times \in \Sigma$ is the reject state.

Remarks : To interpret the formal definition of a Turing machine, one imagines a control with a head or pointer on an infinite tape, in one direction. On the control, one could indicate the current state of the Turing machine. Initially, the Turing machine is on the start state, s_0. The input $w = w_1 w_2 w_3 ... w_n$ is a string of elements from the alphabet Λ. First, w is placed at the left most end of the infinite tape. Since a string is finite, $\sqcup$ blank symbols fill in the remaining entries. Note that since $\sqcup \notin \Lambda$, the first blank symbol indicates the end of the string. At the beginning of the computation, the Turing machine head is on the first symbol of the string. The Turing machine is on the start state s_0 and has its head on the first symbol w_1. One can represent this as the ordered pair (s_0, w_1). Using the transition function, one finds that $\delta(s_0, w_1) = (s, \alpha, D)$, where $s \in \Sigma$, $\alpha \in \Gamma$, and $D \in \{\text{L,R}\}$. To interpret this, one would change the state to s, replace w_1 by α, and move D which is either Left or Right. The next action would be similarly determined by the transition function's value at $\delta(s, w)$, where w is whatever symbol the Turing machine head is currently on. Note that if the head is already at the left most end of the infinite tape, and if the transition function indicates a left movement, the Turing machine would simply change states and write on the tape according to the function but would remain in the same place. The Turing machine will stop if it ever enters the accept $s_\star$ or reject $s_\times$ state. The Turing machine would return the output "Accept" or "Reject". If a Turing machine enters either of these two states the Turing machine is said to halt on the particular input. The remaining case is that the Turing machine will never enter the accept or reject state and would continue forever. In this case, the Turing machine is said to Loop on the particular input.

Remarks : At any particular step, one can indicate the condition or status of the Turing machine by three pieces of information: the string α before the head, the current state s, and the string β which the head is on and follows the head. These three pieces of information determine the configuration, which can be represented as $\alpha\ s\ \beta$. Note that if the Turing machine happens to be at the left most end of the infinite tape, the configuration would be written $s\beta$. As an example, suppose the tape alphabet of a Turing machine T is $\Gamma = \{0, 1, \sqcup, \times, \circ\}$. α the string preceding the head is $011 \sqcup 11110\circ$, the current states is s, and $\beta = 00 \times \circ \times \times \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup ....$ Then the configuration of the Turing machine at this moment is $011 \sqcup 11110 \circ\ \ s\ 00 \times \circ \times \times \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup ....$ One says that a configuration C_1 yields a configuration C_2 if the Turing machine T can go from C_1 to C_2 is just one step. To more precisely define a Turing machine's computation, let $a, b, c \in \Gamma$, the tape alphabet. Let the front string $\alpha = \alpha' \cdot a$, where α' is some string. This notation means that α is the string obtained by adding the symbol a to the end of another string α'. Let $\beta = b \cdot \beta'$, where β' is some string. This notation, similarly, represents the string obtained by adding the symbol b in front of the string β'. Suppose the current configuration of T is $C_1 = \alpha\ s\ \beta = \alpha' \cdot a\ s\ b \cdot \beta$. Suppose the $\delta(s, b) = (s', c, L)$, then C_1 yields $C_2 = \alpha'\ s'\ ac \cdot \beta'$. Had the transition function been $\delta(s, b) = (s', c, R)$, the next configuration would have been $\alpha' \cdot ac\ s'\ \beta'$. Again, one makes the convention that if the Turing machine is at the left most end of the infinite tape, the head stays put even if the transition function indicates left.

Definition 3.2 : Let $T = \{\Sigma, \Lambda, \Gamma, s_0, s_\star, s_\times\}$. Let w be the input string. The start configuration of the Turing machine is $s_0\ w$. T is said to be in an accept configuration if the state of the configuration is $s_\star$. T is in a reject configuration if the state of the Turing machine is in $s_\times$. Both of these configurations are known as halting configurations.

Definition 3.3 : Let $T = \{\Sigma, \Lambda, \Gamma, s_0, s_\star, s_\times\}$ and w be an input string. T accepts w if and only if there is

a sequence of configurations $C_1, ..., C_n$ such that C_1 is the start configuration of T on this input, each C_k yields C_{k+1}, and C_n is an accept configuration. One can similarly define what it means for T to reject w by changing C_n to a reject configuration. T loops on w if T neither accepts nor rejects w.

Remarks : A Turing machine can be described using a Formal Description, an Implementation Description, or a High-level Description. The formal description is the 7-tuple with the set of states, alphabets, transition function, etc. all properly and fully given. The implementation description uses a natural language (English) to describe how the Turing machine moves its head and how it writes on the tape. The High Level Description uses a natural language to simply describe the Turing machine. Sometimes, this paper will use High Level descriptions. To justify this, note that with the increasing complexity of problems, more states and symbols (in the tape alphabet) will be needed (to create a Turing machine whose function is conceptually easy to see). The domain of the transition function is an ordered pair, therefore, to define the transition function formally, one would need to define the image of all $|\Sigma|$ $|\Gamma|$ number of ordered pairs in the domain. However, one feels that the High-Level description of a Turing machine actually corresponds to an existent Turing machine, in that High-level description will describe a purely mechanical procedure to solve some problem. That is, if a human being, devoid of all ingenuity, could follow certain directions in the description to infallibly solve a problem, then a Turing machine can be constructed to mimic this procedure; albeit, one must tediously and meticulously write such a Turing machine.

Definition 3.4 : Let T be a Turing machine. The set of all strings that T accepts is called the language recognized by T, denoted $L(T)$. A language is Turing-Recognizable if there exists a Turing machine that recognized it.

Remarks : Often for practical purposes, it can be extremely difficult to determine whether a Turing machine loops on an input or takes a very long time to halt. To remedy this problem, one can design a Turing machine in such a way that it is known to halt on all inputs.

Definition 3.5 : A Decider is a Turing machine that halts on all inputs. If a decider recognizes some language, then one says that it decides the language. A language is decidable if some Turing machine decides it.

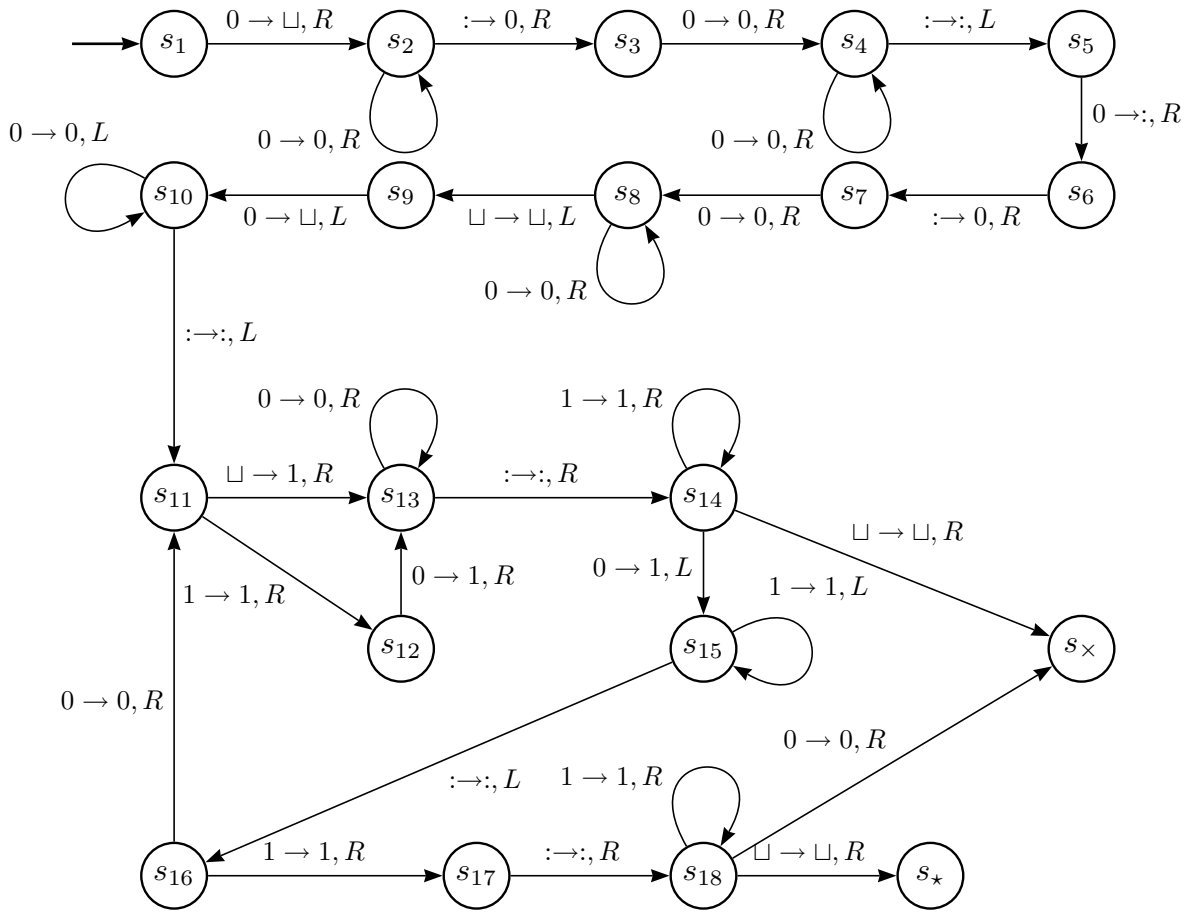
Remarks : Like for finite automata, a concise way of visually representing a Turing machine is with a state diagram. A Turing machine diagram is very similar to finite automata's diagram. The state state s_0 is denoted by an arrow that points in from no other states. Between states there are arrows labeled in the form $a \to b, D$, where $a, b \in \Gamma$ and $D \in \{L, R\}$. Suppose between state s_{13} and s_4 there is the arrow labeled $a \to b, R$. This indicates that when the machine is one state s_{13} and read the input symbol a, it writes b in the spot of a and then moves right. The Turing machine accepts a particular input if it ends at an accept state. This paper makes the convention, for simplicity, that if on a given state, one does not see an arrow corresponding to a particular input symbol, the Turing machine goes to the reject state. Therefore, in a Turing machine state diagram, the reject state is tacitly assumed; however, sometimes for emphasis some arrows leading to reject states are added.

Theorem 3.6 : Let $\Lambda = \{0, 1, :\}$ be an alphabet. Let A be a language over Λ defined by $A = \{0^i : 0^j : 0^k \mid i + j = k \text{ and } i, j, k > 0\}$, then A is a Turing recognizable language and a decidable langauge.

Proof : One shall construct a Turing machine that decides the language A. The construction proceeds as follows: First, one checks that there is at least one 0 to begin with. One writes over this with a blank symbol $\sqcup$ to denote the beginning. Then one skips all subsequent zero, and looks for the : symbol. This Turing

machine wants to combine $0^i : 0^j : 0^k$ into $0^{i+j} : 0^k$, so it writes over the : with a 0 and moves right. It checks that there is at least one 0 following the symbol and skips over all subsequent 0. When it reaches the second : symbol, it moves left and changes that 0 into a : (to account for the extra zero 0 created by the first replacement of : by a 0) and then moves right. It encounters the original :, so it writes it over by a zero. Checks that there is at least one zero following to the right. Skips over all subsequent zeros. When it reaches the blank symbol $\sqcup$ it has reach the end of the input. It moves left and replaces the rightmost 0 by $\sqcup$ to account for the extra 0 produced when replacing :. This finishes the check to see that the input is actually in the right form, and it puts the input into the form $0^{i+j} : 0^k$. Now one moves the reading head to the very beginning. Note that the Turing machine knows where the beginning is because one placed the $\sqcup$ at the very beginning. It changes it to a 1. Skip all intermediate 0, goes past the : symbol, and replace one 0 by a 1. Goes back to the beginning and "crosses" the next zero out with a 1 and moves to the other side of the : and crosses out a 0 with a 1. If at any point, the machine crosses out a 0 but can not find a 0 on the right side to cross out, the machine rejects the input. If after crossing out a 0 from the right side, the machine can not find a 0 on the left side, it goes back to the right side. If it find a 0 on the right side, it rejects. If it only finds $\sqcup$ symbols, as the only non-1 symbols, then it would accept. Note that this machine terminates on all input because this Turing machine never writes over blank symbols and the first time the machine encounters the blank symbol it goes back left. The second time it encounter the blank symbol it either accepts or rejects (depending on certain conditions). It could also reject if simply $i + j \neq k$. Because all input strings are finite, this machine will either accept or reject a string because it is in the wrong form, because $i + j \neq k$, or because it will hit a blank symbol a second time (representing a reject or accept).

To be more precise, this paper gives the state diagram of this Turing machine below:



Note that although, only two states point to the reject state, by convention, the Turing machine goes to

the reject state whenever the arrows that correspond to a particualar state and symbol does not appear on the diagram. This occurs in the early states, where one is merely checking if the input is in the appropriate form.

The formal description of this Turing machine is $\{\{q_1, ..., q_{18}, q_\star, q_\times\}, \{0, 1, :\}, \{0, 1, :, \sqcup\}, \delta, q_1, q_\star, q_\times\}$, where the transition function δ given by $\delta =$
$\{(q_1, 0, q_2, \sqcup, R), (q_1, 1, q_\times, 1, R), (q_1, :, q_\times, :, R), (q_1, \sqcup, q_\times, \sqcup, R),$
$(q_2, 0, q_2, 0, R), (q_2, 1, q_\times, 1, R), (q_2, :, q_3, 0, R), (q_2, \sqcup, q_\times, \sqcup, R),$
$(q_3, 0, q_4, 0, R), (q_3, 1, q_\times, 1, R), (q_3, :, q_\times, :, R), (q_3, \sqcup, q_\times, \sqcup, R),$
$(q_4, 0, q_4, 0, R), (q_4, 1, q_\times, 1, R), (q_4, :, q_5, :, L), (q_4, \sqcup, q_\times, \sqcup, R),$
$(q_5, 0, q_6, :, R), (q_5, 1, q_\times, 1, R), (q_5, :, q_\times, :, R), (q_5, \sqcup, q_\times, \sqcup, R),$
$(q_6, 0, q_\times, 0, R), (q_6, 1, q_\times, 1, R), (q_6, :, q_7, 0, R), (q_6, \sqcup, q_\times, \sqcup, R),$
$(q_7, 0, q_8, 0, R), (q_7, 1, q_\times, 1, R), (q_7, :, q_\times, :, R), (q_7, \sqcup, q_\times, \sqcup, R),$
$(q_8, 0, q_8, 0, R), (q_8, 1, q_\times, 1, R), (q_8, :, q_\times, :, R), (q_8, \sqcup, q_9, \sqcup, L),$
$(q_9, 0, q_{10}, \sqcup, L), (q_9, 1, q_\times, 1, R), (q_9, :, q_\times, :, R), (q_9, \sqcup, q_\times, \sqcup, R)$
$(q_{10}, 0, q_{10}, 0, L), (q_{10}, 1, q_\times, 1, R), (q_{10}, :, q_{11}, :, L), (q_{10}, \sqcup, q_\times, \sqcup, R),$
$(q_{11}, 0, q_{11}, 0, L), (q_{11}, 1, q_{12}, 1, R), (q_{11}, :, q_\times, :, R), (q_{11}, \sqcup, q_{13}, 1, R),$
$(q_{12}, 0, q_{13}, 1, R), (q_{12}, 1, q_\times, 1, R), (q_{12}, :, q_\times, :, R), (q_{12}, \sqcup, q_\times, \sqcup, R)$
$(q_{13}, 0, q_{13}, 0, R), (q_{13}, 1, q_\times, 1, R), (q_{13}, :, q_{14}, :, R), (q_{13}, \sqcup, q_\times, \sqcup, R),$
$(q_{14}, 0, q_{15}, 1, L), (q_{14}, 1, q_{14}, 1, R), (q_{14}, :, q_\times, :, R), (q_{14}, \sqcup, q_\times, \sqcup, R),$
$(q_{15}, 0, q_\times, 0, R), (q_{15}, 1, q_{15}, 1, L), (q_{15}, :, q_{16}, :, L), (q_{15}, \sqcup, q_\times, \sqcup, R),$
$(q_{16}, 0, q_{11}, 0, R), (q_{16}, 1, q_{17}, 1, R), (q_{16}, :, q_\times, :, R), (q_{16}, \sqcup, q_\times, \sqcup, R),$
$(q_{17}, 0, q_\times, 0, R), (q_{17}, 1, q_\times, 1, R), (q_{17}, :, q_{18}, :, R), (q_{17}, \sqcup, q_\times, \sqcup, R)$
$(q_{18}, 0, q_\times, 0, R), (q_{18}, 1, q_{18}, 1, R), (q_1, :, q_\times, :, R), (q_{18}, \sqcup, q_\star, \sqcup, R)$
$(q_\star, 0, q_\star, 0, R), (q_\star, 1, q_\star, 1, R), (q_\star, :, q_\star, :, R), (q_\star, \sqcup, q_\star, \sqcup, R),$
$(q_\times, 0, q_\times, 0, R), (q_\times, 1, q_\times, 1, R), (q_\times, :, q_\times, :, R), (q_\times, \sqcup, q_\times, \sqcup, R)\}$ ■

Definition 3.7 : A Turing machine is a 6-tuple $(\Sigma, \Lambda, \Gamma, \delta, s_0, F)$ where Σ is a finite set of objects, called states, Λ is a finite set (called the Alphabet) of objects such that $\sqcup \notin \Lambda$ (the blank symbol), Γ is a finite set (called the Tape Alphabet) such that $\sqcup \in \Gamma$ and $\Lambda \subset \Gamma$, $\delta : \Sigma \times \Gamma \to \Sigma \times \Gamma \times \{L, R\}$ called the transition function, $s_0 \in \Sigma$ is the start state, and $F \subset \Sigma$ is the set of halting states.

Remarks : Everything is interpreted the same as the first definition of a Turing machine, except here one permits more than just two halting states.

Theorem 3.8 : The Turing machine in Definition 3.1 is equivalent to the Turing machine is Definition 3.7.

Proof : It is clear that the first Turing machine has an equivalent Turing machine of the second kind. Simply let $F = \{s_\star, s_\times\}$.

A Turing machine of the second kind can be converted into a Turing machine of the first kind by designating one element of F to be $s_\star$. Then let every other element of F be renamed $s_\star$. That is, in the transition function, whenever, one sees a state $s \in F$, then one changes it to $s_\star$. Now if for a particular string the Turing machine of the second kind halts, then it lands on a state $s \in F$, but this means that this same input caused the new converted Turing machine of the first kind to land on $s_\star$; therefore, the new Turing machine accepts this string as well. The two Turing machines accept the same language. ■.

Remarks : For conceptual reason and ease, one sometimes would like a Turing machine to stay put (represented as S) after changing states and rewriting an entry. The next theorem shows such a Turing machine

is equivalent to the Turing machine in Definition 3.1.

Theorem 3.9 : A Turing machine that is the same as the Turing machine in Definition 3.1, except $\delta : \Sigma \times \Gamma \to \Sigma \times \Gamma \times \{L, R, S\}$ is equivalent to the Turing machine in Definition 3.1.

Proof : It is clear that a Turing machine of the first kind is a Turing machine of this new type. Simply, the Turing machine of the first kind has a transition function that never uses S.

To show that the new Turing machine is equivalent to the first Turing machine, note that the new transition function is still a set of 5-tuples. Consider the set $\mathfrak{A}$ all ordered tuple of the form $(q, \alpha, q', \beta, S)$. Add the following new states q_α for each q and α that make up the first two entry of the same 5-tuple in $\mathfrak{A}$. Then remove every 5-tuple in $\mathfrak{A}$ from the transition function, and replace each deleted $(q, \alpha, q', \beta, S)$ with $(q, \alpha, q_\alpha, \beta, R)$ and $(q_\alpha, \beta, q', \beta, L)$. This alteration removes all the 5-tuples that contains the stay put instruction. This Turing machine of the first kind clearly imitates the stay-put Turing machine since it merely moves right and then move back left. ■

Remarks : Because one has shown that all three Turing machines presented thus far are equivalent, this paper could have used any as the "basic" Turing machine. Definition 3.1 played this role because it appears simplier; however, from hence forth, the term Turing machine will alway mean the last Stay-put Turing machine (of Theorem 3.9). This is for conceptual reasons.

Definition 3.10 : A Multitape Turing machine with k tapes is a 7 tuple $(\Sigma, \Lambda, \Gamma, \delta, s_0, s_\star, s_\times)$ where Σ is a finite set of objects called states, Λ is a finite set (called the Alphabet) of objects such that $\sqcup \notin \Lambda$ (the blank symbol), Γ is a finite set (called the Tape Alphabet) such that $\sqcup \in \Gamma$ and $\Lambda \subset \Gamma$, $\delta : \Sigma \times \Gamma^k \to \Sigma \times \Gamma^k \times \{\mathrm{L,R,S}\}^k$ called the transition function, $s_0 \in \Sigma$ is the start state, $s_\star \in \Sigma$ is the accept state, and $s_\times \in \Sigma$ is the reject state.

Remarks : The interpretation of this definition is that the multitape Turing machine is a Turing machine with several tapes. The first tape contains the input and the other tapes intially contain only blank symbols. If the Turing machine is currently in state s and the pointers are on $b_1, b_2, b_3, ..., b_k$ of the first, second, ..., k^{th} tape, respectively, and $\delta(s, b_1, ..., b_k) = (s', c_1, ..., c_k, D_1, ..., D_k)$ where $D_i \in \{L, R, S\}$, then the Turing machine changes to state s', writes c_i on the i^{th} tape, and move that pointer in the D_i direction.

On the state diagram of a multitape Turing machine with n tapes, suppose there is an arrow from state s to q, then the arrow would take the form $a_1, ..., a_n \to b_1...b_n, D_1, ..., D_n$, where $a_i, b_i \in \Gamma$ and $D_i \in \{L, R, S\}$ for $1 \leq i \leq n$. This arrow means $\delta(s, a_1, ..., a_n) = (q, b_1, ..., b_n, D_1, ..., D_n)$.

Theorem 3.11 : Every multitape Turing machine is equivalent to a Turing machine.

Proof : Suppose $M = (\Sigma^M, \Lambda^M, \Gamma^M, \delta^M, s_0^M, s_\star^M, s_\times^M)$ is a multitape Turing machine with k tapes. Let $T = (\Sigma^T, \Lambda^T, \Gamma^T, \delta^T, s_0^T, s_\star^T, s_\times^T)$ be the Turing machine that should imitate M. Let $\Sigma^T = \Sigma^M$. As usual let $\Sigma^T \subset \Gamma^T$ and $\sqcup \in \Gamma^T$. However, let the following symbol $| \in \Gamma^T$, let $\Gamma^M \subset \Gamma^T$ and for each $\alpha \in \Gamma^M$, $\bar{\alpha} \in \Gamma^T$. One wish to store the content of all k tapes of M onto the single tape of T. The symbol "|" seperate the string on seperate tapes and if on the i^{th} tape, the pointer sits on the symbol $\alpha \in \Gamma^M$, then in T, on the substring that represent the i^{th} tape, one replaces the symbol α by $\bar{\alpha}$ to denote the pointer currently sits there.

On input $w = w_1...w_n$, the Turing machine T rewrites its tapes into the form $|\bar{w_1}w_2...w_n|\bar{\sqcup}|\bar{\sqcup}|...|\bar{\sqcup}\sqcup\sqcup...$, where there are $k-1$ $\bar{\sqcup}$. This represents M having its first points on the input w, and all its other pointers on blank symbols. Since this procedure is purely mechanical, one is safe to assume such a Turing machine can be written. Next T scans this string starting from the first | to the last, looking for what symbols are

under the bar. It then consults the transition function of M to see how to change these symbols and where to move the bars. If the pointer move the bar right onto |, then it adds on a $\bar{\sqcup}$ to represent M moving onto a blank symbol at the end of a particular tape. If the pointer moves left onto the |, then it stays put. Again this process is mechanical, and one can construct such a Turing machine by creating the appropriate trasition function δ^T which depend on δ^M and additional symbols to Γ^T, if necessary. It is clear that T will reach an accept state for a particular input if and only if M accepts that particular input, as well. ■

Remarks : Historically, various models of computation (with certain restrictions such that as the ability to do only finite amount of "work" per step) have been shown to be equivalent to each other. Without details, Turing machines, multitape Turing machine, nondeterministic Turing machine, unlimited register machine, λ-calculus, general recursive functions, and several other models have been proven to be equivalent.

Furthermore, a totally non-mathematical statement known as the Church-Turing Thesis essentially asserts that the intuitive notion of algorithms is the same as a Turing machine. One can even define an algorithm in term of a Turing machine. If one accepts the Church-Turing Thesis, one could justify the various instances in the above proofs where the paper assumes there is a Turing machine which does precisely what was described and believed to be an algorithmic process.

In light of the Church-Turing Thesis, for several of the following next results, this paper will outline the Turing machine algorithm that produces a certain result but will not give the state diagram or formal description. The description will describe a definite algorithmic process, and one could write out an explicit Turing machine given time and care.

Often a Turing machine may be used to solve problems about various objects, such as polynomials, graphs, finite automaton, and even other Turing machines. However, the input of a Turing machine is always a string in some language. Therefore, often one needs to make an encoding to represent that object as a string. If O is some object in consideration, the paper shall use the symbol $\langle 0 \rangle$ to be the encoding string. This paper shall not be concerned about how the object is encoded because one could always make a Turing machine translate one encoding into an encoding of another form. The important fact is that an object can be encoded into any language with at least two symbols. However, during the construction of the Universal Turing machine, this paper will give an explicit encoding of Turing machines as strings in the language of the Universal Turing machine.

Section 4: Decidability in Languages

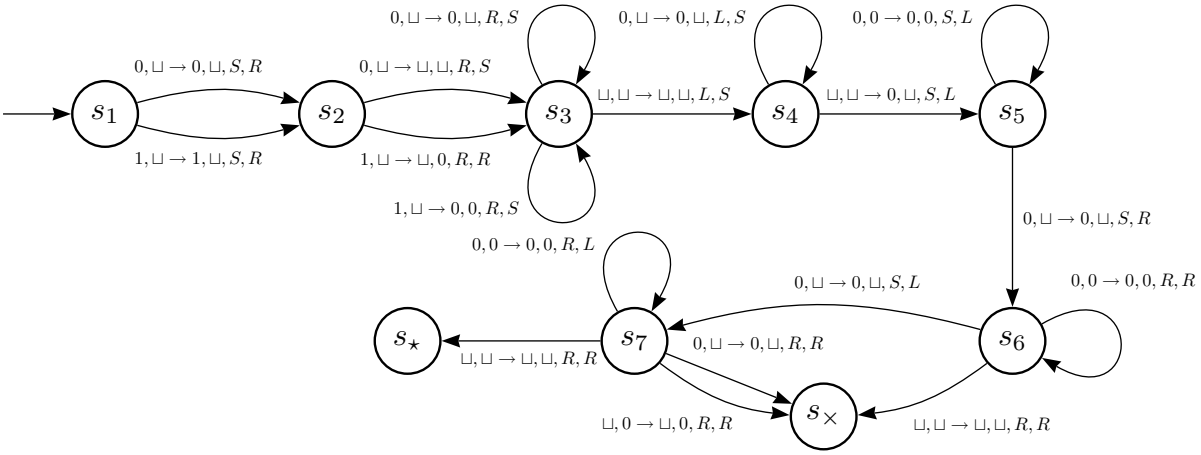
Theorem 4.1 Let $\Lambda = \{0, 1\}$. Let J be a language over Λ defined as follows:

$$J = \{w \mid w \text{ contains equal numbers of 0 and 1}\}$$

This language is decidable.

Proof : One will construct a decider Turing machine D that recognizes this language. D will be a two tape Turing machine. Initially, it will have the input in the first tape. The second tape will contain all blank symbols. First, it will move the head of the second tape to the second blank symbol $\sqcup$. The first blank symbol will indicate the beginning of the second tape. On the first input symbol of tape 1, if it is a zero, the machine will write it over by a $\sqcup$ (to indicate the beginning of tape 1) and move right. If it is 1, then D will write $\sqcup$ (to indicate beginning of tape 1) and write a 0 on tape 2 and move right on both tapes. Tape 2 is used to count the number of 1 that occurs. For all other symbols, if the tape 1 symbol in the reading frame is 0, the reading pointer moves right and tape 2 does nothing. If the input symbol in the reading frame of tape 1 is 1, then D writes over 1 with a 0 and in tape 2, writes over a $\sqcup$ with a 0, and both tapes move right. This process continues until, the input string is over (as indicated by a $\sqcup$). At this point tape 1 and 2 have all zeros. If the number of 0 in tape 1 is two times the number of 0 on tape 2, then the input had equal number of 0 and 1. It would accept otherwise reject. To check this, tape 1 must move back to the beginning. When it hits the $\sqcup$ that D placed at the beginning, it changes it to 0. Then tape 1 stays put. Then D moves tape 2 back to the beginning. When it hits $\sqcup$, tape 2 moves right to the first 0. Now both tape's reading head are on zeros. As long as both reading head are 0, both strings move right. When tape 2 reaches $\sqcup$ at the end, if tape 1 is also on $\sqcup$, then this means the input had only 1. Reject. Otherwise, tape 1 is on a zero, so move tape 2 left. Now both tape 1 and tape 2 heads are on 0. Move tape 1 right, and move tape 2 left. So long as both reading heads are on 0, move tape 1 right and move tape 2 left. If tape 1 has a 0 and tape 2 has $\sqcup$, then reject because the input had more zeros than 1. If tape 1 has $\sqcup$ and tape 2 has 0, then reject because there were more 1. If both reaches $\sqcup$ at the same time accept because the number of 1 and 0 were the same. D halts on all input because it never changes the $\sqcup$ at the end into any other symbol. The first time tape 1 reaches $\sqcup$, it moves back. Then D keeps moving right and when it hits $\sqcup$ again it accepts or rejects (unless something else happens before that which causes D to reject).

Since this Turing machine is quite simple, the state diagram is provided below:



The formal description of this Turing machine is $\{\{s_1, ..., s_7, s_\star, s_\times\}, \{0, 1\}, \{0, 1, \sqcup\}, \delta, s_1, s_\star, s_\times\}$, where the transition function δ can easily be read off the diagram. This paper does not list the set of 8-tuple that make δ because there are $|\Sigma|\ |\Gamma|^2 = 9 \cdot 3^2 = 81$ such tuples. ■

Remarks : Note that J is the same language in Theorem 2.26. So there are decidable langauges that are

non-regular.

Theorem 4.2 : Every regular language is a decidable language.

Proof : Let A be a regular language recognized by a deterministic finite automaton $D' = \{\Sigma', \Lambda', \delta', s_0', F\}$. To prove, the decidability of this language, one must construct a Decider $D = \{\Sigma, \Lambda, \Gamma, \delta, s_0, s_\star, s_\times\}$ for this language.

D will be a Turing machine that only moves right. When D reads a particular input symbol, it will go to the state (of the Turing machine corresponding to the state of the finite automaton) indicated by the transition function δ'. The end of the input string is indicated by $\sqcup$. Therefore, when D reads $\sqcup$, if the current state of the Turing machine is a state in F, the Turing machine goes to $s_\star$ (accepts). If it reads $\sqcup$ and the current state is in $\Sigma'\backslash F$, then the Turing machine goes to state $s_\times$. This is a decider (i.e. halts on all inputs) because this machine only moves right. When the reading head reaches $\sqcup$, it must decide to go to $s_\star$ or $s_\times$. Because a string has only a finite number of symbols, it must eventually reach the $\sqcup$ at the end.

Constuct D as follows. Let $s_\star$ and $s_\times$ represent the accept and reject state. Let $\Sigma = \Sigma' \cup \{s_\star, s_\times\}$. Let $\Lambda = \Lambda'$. Let $\sqcup$ be the blank symbol. Then let $\Gamma = \Lambda \cup \{\sqcup\}$. The start state is s_0. The accept state is $s_\star$, and the reject state is $s_\times$. The transition function is

$$\delta(s,a) = \begin{cases} (\delta'(s,a), a, R) & \text{if } s \in \Sigma \text{ and } a \neq \sqcup \\ (s_\star, a, R) & \text{if } s \in F \text{ and } a = \sqcup \\ (s_\times, a, R) & \text{if } s \in \Sigma\backslash F \text{ and } a = \sqcup \\ (s_\star, a, R) & \text{if } s = s_\star \\ (s_\times, a, R) & \text{if } s = s_\times \end{cases}$$

■

Theorem 4.3 : Let $A = \{\langle B, w\rangle \mid B \text{ is a deterministic finite automaton that accepts } w\}$. (Note $\langle B, w\rangle$ is an encoding of B and w). A is a decidable language.

Proof : From the encoding of B, have a Turing machine D simulate B on w. There is certainly a Turing machine that can imitate an arbitrary finte automaton. (In fact, later this paper will construct the Universal Turing which imitates arbitrary Turing machines.) First, one should make a Turing machine that checks if the input string is actually a finite automaton encoding. Run the simulator Turing machine on w. If the simulation indicates that B accepts w, then D accepts $\langle B, w\rangle$. If the simulation indicates that B rejects w, then D rejects. D certainly halts on all input. First, if the input is not in the form of an encoding of $\langle B, w\rangle$, the Turing machine would have thrown it out during checking. If it is an actually finite automaton and a appropriate string, then D would halt since all finite automaton accept or reject any string in a finite number of steps. ■

Theorem 4.4 : Let $E = \{\langle B\rangle \mid B \text{ is a finite automaton and } L(B) = \emptyset\}$. E is decidable.

Proof : This will be a High-Level Description of a clearly algorithmic procedure. If one accept the Church-Turing Thesis, then there is a Turing machine that represents this process.

First, one would mark the start state of B. Then mark all states that have arrows coming from the start state of B. Recursively, mark all states that have arrows coming from already marked states. Repeat, this procedure until no new marked states are generated. Then check to see if an accept state is marked. If an accept state is marked then there is a string that can get there. Accept. If no accept states are marked,

reject. ■

Theorem 4.5 : Let $Q = \{\langle A, B\rangle \mid A \text{ and } B \text{ are finite automata and } L(A) = L(B)\}$. Q is decidable.

Proof : One seeks to construct D a decider for Q. Given the encoding of A and B, one can construct a finite automaton C that recognizes $L(A)\Delta L(B)$. The Turing machine D can construct C by the same algorithmic constructions used in the proof of the closure of regular langauges under various operations in Section 2. By Theorem 4.4, there is a Turing machine T that decides whether the langauge of a finite automaton is empty. As a sub-Turing machine of D, run T on the newly constructed C. If T accept $\langle C\rangle$, then D accepts $\langle A, B\rangle$ because T accepting C means that the $L(C) = L(A)\Delta L(B) = \emptyset$. This only happen if $L(A) = L(B)$. If T rejects, then D would also reject. ■

Theorem 4.6 : The set of all Turing machine is countable.

Proof : Note, that the alphabet of each Turing machine must be finite. Other than how one names the symbols, all alphabets of the same cardinality are the same. Hence alphabets are determined by a natural number n which is the number of symbols in that alphabet. Therefore, the set of all alphabets has cardinality $\aleph_0$. Similarly the set of states and the tape alphabet are finite sets determined by the number of elements in them; so they are of cardinality $\aleph_0$. For a fixed (and always finite) Σ and Γ, the cardinality of the domain of δ which is $\Sigma \times \Gamma$ is $|\Sigma|\ |\Gamma|$. The cardinality of the range, which is $\Sigma \times \Gamma \times \{L, R, S\}$, is $|\Sigma|\ |\Gamma| \cdot 3$. Since both the range and domain are finite, the number of such function $\delta : \Sigma \times \Gamma \to \Sigma \times \Gamma \times \{L, R, S\}$ is $(3|\Sigma|\ |\Gamma|)^{|\Sigma|\ |\Gamma|}$ by Theorem 1.11. Let $\Delta_{\Sigma,\Gamma} = \{\delta \mid \delta : \Sigma \times \Gamma \to \Sigma \times \Gamma \times \{L, R, S\}\}$. Let Δ be the set of all possible transition function for every possible Turing machine. One then has

$$\Delta = \bigcup_{\Sigma}\bigcup_{\Gamma} \Delta_{\Sigma,\Gamma}$$

Where the unions are taken over all possible Σ and Γ for any possible Turing machine. Since the set of all set of states Σ and the set of tape alphabet Γ are countable, one sees that Δ is a countable union of countable sets; therefore, Δ is countable by Theorem 1.9.

A Turing machine is the 7-tuple $(\Sigma, \Lambda, \Gamma, s_0, s_\star, s_\times)$. The set of the set of states, the set of all alphabets, and the set of all tape alphabet are countable. $s_0, s_\star,$ and $s_\times$ are elements of the set of state. For any particular Turing machine, there are only a finite number of states; hence, only finite choices of states would make sense in the 7-tuple. But as an added liberty, let allow $s_0, s_\times, s_\star$ come from the union of all possible set of states. Since each set of state is finite, this union, call it U can at most be countable. So there are countable choices for these three values. The set of all ordered 7-tuple is the cartesian product of the set of set of states with the set of alphabets with the the set of tape alphabets, with δ (the set of all possible transition functions), and U^3. All of these sets are countable, so by Theorem 1.10, the finite cartesian product is countable. Note that only a subset of this cartesian product corresponds to a Turing machine, since in a given tuple, the transition function from δ or the states from U may not correspond to the particular $\Sigma, \Lambda,$ and Γ. However, subset of countably infinite sets can at most be countably infinite. ■

Theorem 4.7 : The set of all languages over an alphabet is uncountable.

Proof : First, given an alphabet, the set A of all (finite) strings is countable. Let n be the cardinality of the alphabet. Let A_i be the set of all strings of length i. The cardinality of A_i is n^i, since each place can have n possible symbols. One has that $A = \bigcup_{n\in\mathbb{N}} A_n$ which is a countable union of countable (finite) sets; hence, A is countable. Put a total order on the alphabet. This can be done since the alphabet is just a finite set. List A in lexographical order. Let L be a language over the alphabet. The characteristic sequence of L is a infinite sequence of 0's and 1's such that the i^{th} term of the characteristic sequence is 0 if the i^{th} string in

the lexographical order is not an element of L and 1 if it is. This defines a bijection from the set of languages over the alphabet and the set of sequence of two elements. By Theorem 1.12, we know the set of sequence of two distinct element is uncountable. ■

Theorem 4.8 : There are languages that are not Turing-Recognizable. There are languages that are not decidable.

Proof : By Theorem 2.10, there is an uncountable number of languages over a given alphabet. By Theorem 2.9, there is only a countable number of Turing machine with any alphabet. Since for each Turing machine, there is one language it recognizes, there must be languages that are not recognizable. There are languages that are not decidable since the set of all undecidable languages is a superset of the set of unrecognizable languages. ■

Remark : A Universal Turing machine is a Turing machine that simulates an arbitrary Turing machine when given its description. One possible purpose of the Universal Turing machine is that when given the description or encoding of a Turing machine (in the alphabet of the Universal Turing machine) and the encoding of the input string for that Turing machine (of course in the alphabet of the Universal Turing machine), the Universal Turing machine would return accept if and only if that Turing machine accepts the string.

Theorem 4.9 : There exists a Universal Turing machine $U = \{\Sigma, \Lambda, \Gamma, \delta, s_0, s_\star, s_\times\}$ that when given an encoding $\langle T, w\rangle$ such that T is a Turing machine and w is a string in the alphabet of T, then U accepts $\langle T, w\rangle$ if and only if T accepts w.

Proof : In order to construct in detail an actual Universal Turing machine, one must describe how to encode a Turing machine in some language. The Universal Turing machine to be constructed will have $\Lambda = \{0, +, :\}$ and $\Gamma = \{0, +, :, \sqcup\}$, where $\sqcup$ is the blank symbol. Now let $T = \{\Sigma^T, \Lambda^T, \Gamma^T, \delta^T, s_0^T, s_\star^T, s_\times^T\}$ be a Turing machine and $w = w_1 w_2 ... w_n$ be some string. For simplicity, this Turing machine has at least three distinct states $s_0^T, s_\star^T, s_\times^T$. T may move Left, Right, or Stay put. First, note that $\Sigma^T, \Lambda^T, \Gamma^T$ are all finite sets. Other than the name given to the individual elements of those sets, all sets of states with the same cardinality (number of elements) are essentially the same (and same goes for the Alphabet and the Tape Alphabet). This indicates that one may represent Σ^T with x number of elements by a string of x zeros. Same for Λ^T and Γ^T.

Ultimately one wishes to encode $\langle T, w\rangle$ as a string over $\{0, +, :\}$. So far, one can encode $\Sigma^T, \Lambda^T,$ and Γ^T as the string

$$+ \underbrace{00...00}_{|\Sigma^T| \text{ times}} ++ \underbrace{00...00}_{|\Lambda^T| \text{ times}} ++ \underbrace{00...00}_{|\Gamma^T| \text{ times}} +.....$$

Furthermore, each state in the set of state and each symbol in the alphabet can be represented as a string of 0 in the following way. Put some arbitrary order on the set of alphabet Λ^T and Γ^T such that for all $x, y \in \Lambda^T$ if $x < y$, then $x < y$ as elements of Γ^T. The paper adds an extra convention that all elements of Λ^T be less than elements in $\Gamma^T \backslash \Lambda^T$ and the $\sqcup^T$ is the very last or greatest element. Number the elements according to this order (this can be done since they are finite sets). That number of zeros will indicate that symbol. For example, if x is the 5^{th} symbol, then its encoding will be $\langle x\rangle = 00000$. By the paper's convention, $\langle \sqcup^T\rangle = 00...00$ $|\Gamma^T|$ number of times. Same thing for Σ^T. However, one adds the convention that the start symbol s_0^T is the first state, $s_\star$ is the $(|\Sigma^T| - 1)^{\text{th}}$ state, and $s_\times$ is the last ($|\Sigma^T|^{\text{th}}$ state). The + symbol serves as the delimiter. Later, one will need to encode the directional symbols L, R, and S. The convention is that $\langle L\rangle = 0$, $\langle R\rangle = 00$, and $\langle S\rangle = 000$. Next one must encode the transition function $\delta : \Sigma^T \times \Gamma^T \to \Sigma^T \times^T \Gamma^T \times \{L, R, S\}$. The transition function may be represented as a set of 5-tuples. To add

the information about the transition function onto the encoding string started above, one first use ++ at the beginning and end of the information encoding δ for seperation purposes. Suppose that $(s, a, s', b, R) \in \delta$. One would encode this as $+\langle s\rangle + \langle a\rangle + \langle s'\rangle + \langle b\rangle + \langle R\rangle +$. For simplicity, one allows more than one 5-tuples with the same first two coordinates to be in the encoding of δ. In the actual simulation of T on w by U, U will only use the first 5-tuple that has the required first two coordinates, it is looking for. As an example of δ, suppose one has $(s_0^T, a_2, s_3, a_4, R)$ where s_0 is the start state, a_2 and a_4 are the second and fourth symbol in the alphabet under the arbitrary pre-decided ordering, and s_3 is the third state under the ordering of Σ^T, then its encoding is

$$+0+00+000+0000+00+$$

where the final + indicates the end. So now adding the transition function (in the example) to the encoding string of T, one gets something that looks like the following

$$+\underbrace{00...00}_{|\Sigma^T| \text{ times}}++\underbrace{00...00}_{|\Lambda| \text{ times}}++\underbrace{00...00}_{|\Gamma| \text{ times}}++++0+00+000+0000+00+ \text{... other 5-tuples ...} ++.....$$

The next part of the encoding of T is its start state which by convential is represented by a single 0. Then one has $s_\star$, which is followed by $s_\times$. By convention, $\langle s_\star\rangle$ is $|\Sigma^T| - 1$ zeros and $\langle s_\times\rangle$ is $|\Sigma^T|$ zeros. + again serves as the delimiter. Adding this information on, one has

$$+\underbrace{00...00}_{|\Sigma^T| \text{ times}}++\underbrace{00...00}_{|\Lambda| \text{ times}}++\underbrace{00...00}_{|\Gamma| \text{ times}}++++0+00+000+0000+00+ \text{... other 5-tuples ...} ++$$

$$+0+\underbrace{00...00}_{|\Sigma^T|-1 \text{ times}}+\underbrace{00...00}_{|\Sigma^T| \text{ times}}+.....$$

The final part of the encoding of $\langle T, w\rangle$ is the input string w. If $w = w_1w_2...w_n$, one makes $\langle w\rangle = \langle w_1\rangle : \langle w_2\rangle : ... : \langle w_n\rangle$. The : is the delimiter for different symbols of the imput string. The + will be used to seperate $\langle w\rangle$ from previous part of the encoding. + will also end the encoding. As an example, suppose that $w = a_1^T a_4^T$, where a_1 and a_4 indicate the first and fourth symbol in Λ^T. Then $+\langle w\rangle + = +0 : 0000+$. If one let w be the w in the example, then an example of a complete encoding of $\langle T, w\rangle$ would be

$$+\underbrace{00...00}_{|\Sigma^T| \text{ times}}++\underbrace{00...00}_{|\Lambda| \text{ times}}++\underbrace{00...00}_{|\Gamma| \text{ times}}++++0+00+000+0000+00+ \text{... other 5-tuples ...} ++$$

$$+0+\underbrace{00...00}_{|\Sigma^T|-1 \text{ times}}+\underbrace{00...00}_{|\Sigma^T| \text{ times}}++0:0000+$$

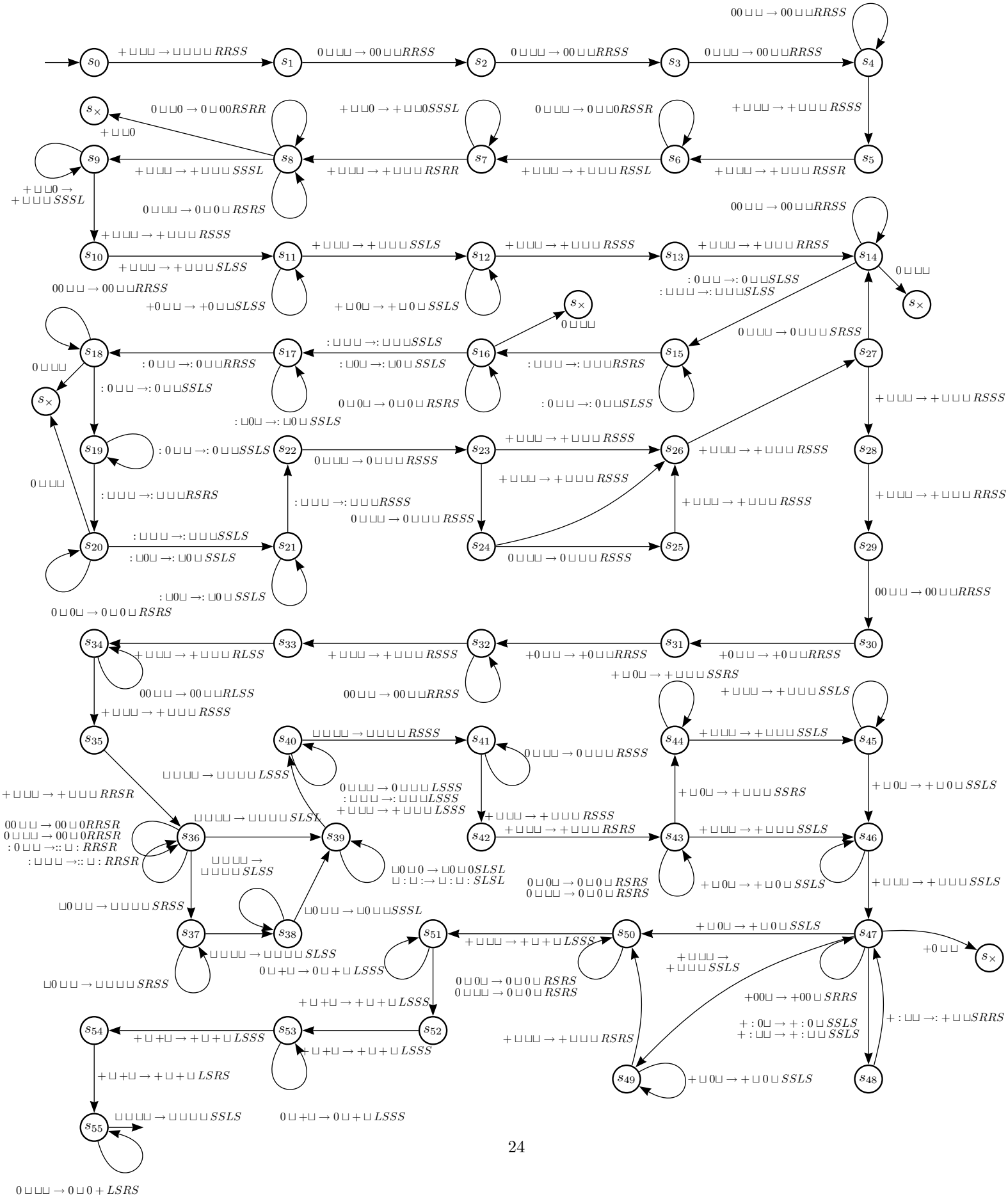
Now for the high level description of U. Note the high-level description will leave out many technical details such as marking the beginning of tapes and exactly how certain things are done. The state diagram of U provided at the end will give the precise details of how U functions.

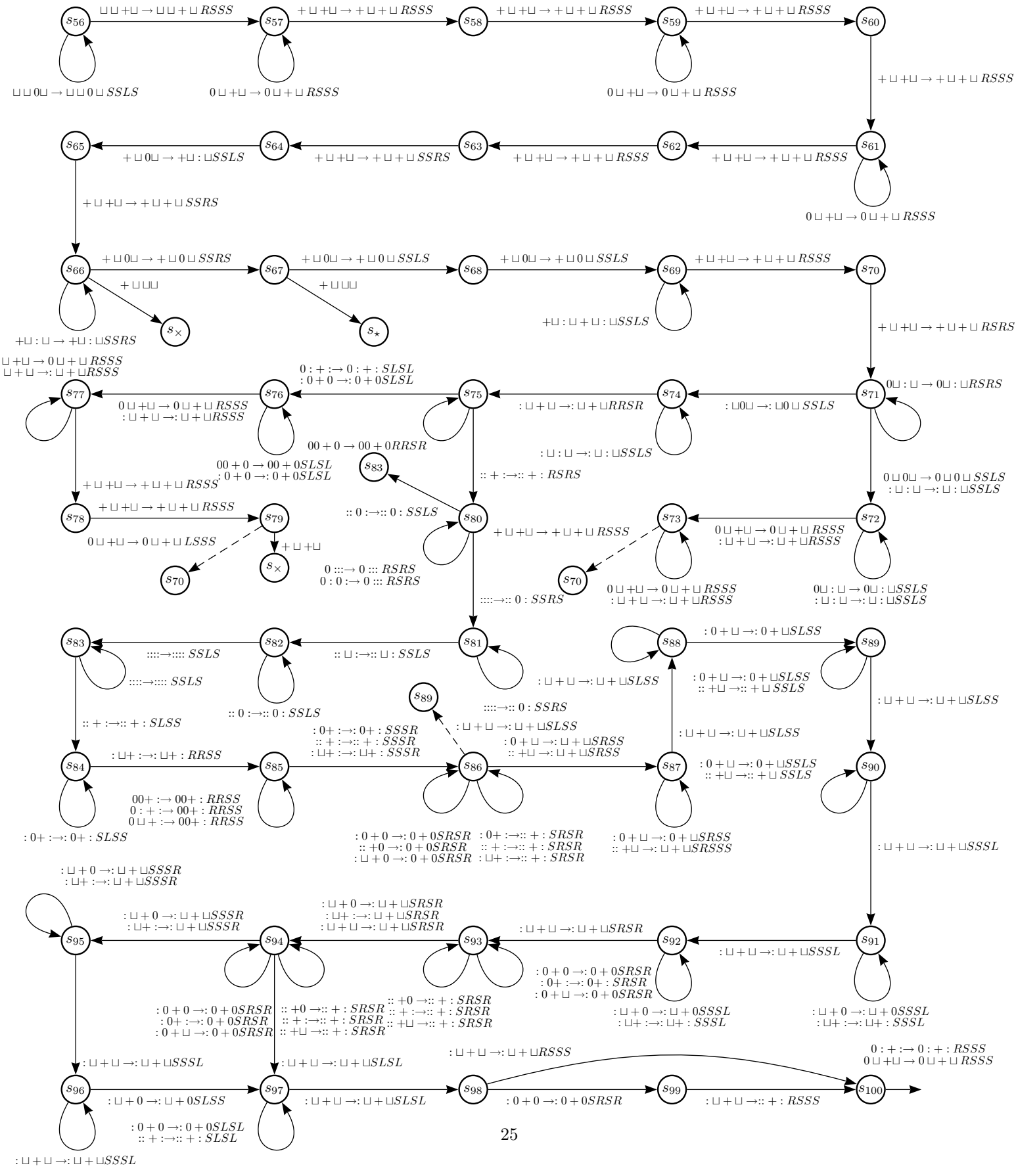
U will be a 4-tape Turing machine. First, U must check whether its given input is even an encoding of a Turing machine and its input. It scans over $\langle\Sigma^T\rangle$ and stores the encoding on tape 2. Then it scans over $\langle\Lambda^T\rangle$ and puts it over tape 4. As it scan $\langle\Gamma^T\rangle$, U will check that $|\Gamma^T| \geq |\Lambda^T|$ and copy Γ^T on to tape 3. U will then erase the content of tape 4 because it is no longer needed. Then U will go into the encoding of δ. It will check each 5-tuple. It will see that the first string of zeros is not bigger than $|\Sigma^T|$ (whose information is written on tape 3) because it is suppose to represent a state of T. Then it checks that the second string of zeros is no bigger than $|\Gamma^T|$, which is written on tape 3, as it is suppose to represent an alphabet symbol. It checks that the thirds string is no larger than $|\Sigma^T|$. It checks that the fourth string is no larger than $|\Gamma^T|$. Then it check the fifth string of zeros is either 0, 00, or 000. U keeps checking 5-tuples until there are no more in the encoding of δ. Next it checks that the input string makes sense. U goes back and copy Λ onto tape 3. Then U goes through the encoding of w to see that the number of 0 that represent the symbols of w do not exceed the number of zeros that represents the input alphabet Λ. As it is checking w, it writes the encoding of w onto tape 2 and tape 4. After it is done, U will need to prepare for the simulation stage. It

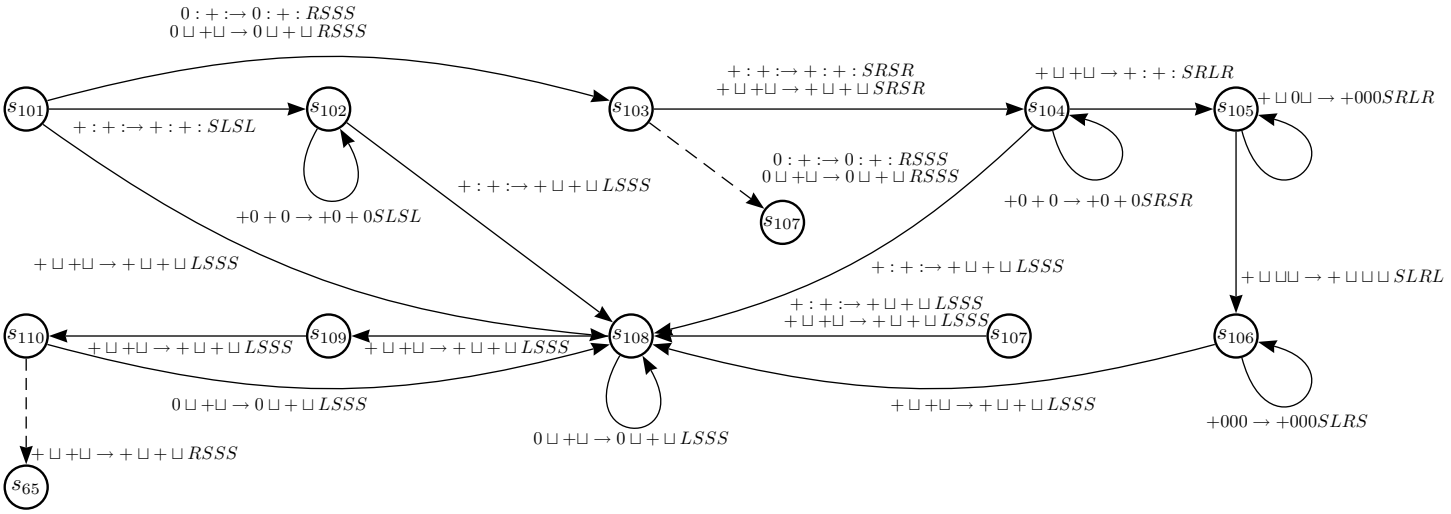
goes back and copy Γ^T on tape 3. Then it copies Σ^T at the end of tape 3, with + as a seperator. One needs $\langle \Gamma^T \rangle$ because it also represents the encoding of $\sqcup$ by convention. Note that U rejects its input string if it fails the check at any point.

The current state (s_0^T at the start of the simulation) is written using : symbols over the encoding of Σ^T on the end of the third tape. The current symbol, which the simulation of T is on, is represented by a $\sqcup$ instead of the + delimiter on the encoding of w on tape 2 and 4. [1] Now for the simulation, first U must check whether it is in the accept or reject state. It uses the encoding of Σ^T to do so. The detail are in the state diagram. If it is in the accept state, U accepts. If it is in the reject state, then U rejects. Note that this reject that represent T rejecting w can be distinguished from merely the encoding failing the check stage simply by running the verfication part of U again. [2] If it is not in either state, then U looks for the current state on the first part of each 5-tuple in the encoding of δ. If it finds a state that matches the current state, it looks at the next symol of that 5-tuple. If it does not match the current symbol on the simulation of T, then it goes back to (2) on the next 5-tuple. If, however, the second coordinate of the encoding of the 5-tuple matches the current symbol of the simulation of T, then U will change its current state which is recorded on tape 3 and write over the encoding of the current symbol of the simulation of T on only tape 2 with the encoding of the symbol in the 4^{th} part of the 5-tuple. Then U uses the copy of the string on tape 4, to write the substring that follows the symbol that was written over at the end of what was just changed on tape 2. Then U copies tape 2 to tape 4. Finally it reads the fifth part of the 5-tuple. If it is $\langle L \rangle$, U moves the $\sqcup$ symbol on tape 2 and 4, which indicates the current reading head position of the simulation of T to the beginning of the encoding of the symbol to the left of the current symbol. Of course, if it is already at the left most end, U does not move at all. Make this change for both tape 2 and 4. Similarly, if it is $\langle R \rangle$, then U moves the head-indicating symbol to the beginning of the symbol to the right of the current symbol. If it moves to $\sqcup$ of U, this means that T moved to a blank symbol as well. U will then change its own $\sqcup$ to the encoding of $\sqcup^T$. Remember that the encoding of $\sqcup^T$ is stored at the beginning of tape 3. If it reads $\langle S \rangle$, U will not moves the head-indicator symbol. When this is done, U moves its own reading head to where it was before part (1). Then U repeats (1) again.

The state diagram of U is given on the next page. Dashed arrows indicate redirection to previous states. Following previously established conventions, the reject states are tacit and only given if the author feels the need for the emphasis.


s_0
$+\sqcup\sqcup\sqcup \to \sqcup\sqcup\sqcup\sqcup RRSS$
s_1
$0\sqcup\sqcup\sqcup \to 00\sqcup\sqcup RRSS$
s_2
$0\sqcup\sqcup\sqcup \to 00\sqcup\sqcup RRSS$
s_3
$0\sqcup\sqcup\sqcup \to 00\sqcup\sqcup RRSS$
s_4
$00\sqcup\sqcup \to 00\sqcup\sqcup RRSS$
$s_\times$
$0\sqcup\sqcup 0 \to 0\sqcup 00RSRR$
$+\sqcup\sqcup 0$
$+\sqcup\sqcup 0 \to +\sqcup\sqcup 0SSSL$
$0\sqcup\sqcup\sqcup \to 0\sqcup\sqcup 0RSSR$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSSS$
s_9
s_8
s_7
s_6
s_5
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup SSSL$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSRR$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSSL$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSSR$
$+\sqcup\sqcup 0 \to$
$+\sqcup\sqcup\sqcup SSSL$
$0\sqcup\sqcup\sqcup \to 0\sqcup 0\sqcup RSRS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSSS$
$00\sqcup\sqcup \to 00\sqcup\sqcup RRSS$
s_{10}
s_{11}
s_{12}
s_{13}
s_{14}
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup SLSS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup SSLS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSSS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RRSS$
$0\sqcup\sqcup\sqcup$
$00\sqcup\sqcup \to 00\sqcup\sqcup RRSS$
$+0\sqcup\sqcup \to +0\sqcup\sqcup SLSS$
$+\sqcup 0\sqcup \to +\sqcup 0\sqcup SSLS$
$:0\sqcup\sqcup \to :0\sqcup\sqcup SLSS$
$:\sqcup\sqcup\sqcup \to :\sqcup\sqcup\sqcup SLSS$
$s_\times$
$0\sqcup\sqcup\sqcup$
$0\sqcup\sqcup\sqcup \to 0\sqcup\sqcup\sqcup SRSS$
s_{18}
s_{17}
s_{16}
s_{15}
s_{27}
$0\sqcup\sqcup\sqcup$
$:0\sqcup\sqcup \to :0\sqcup\sqcup RRSS$
$:\sqcup\sqcup\sqcup \to :\sqcup\sqcup\sqcup SSLS$
$:\sqcup 0\sqcup \to :\sqcup 0\sqcup SSLS$
$:\sqcup\sqcup\sqcup \to :\sqcup\sqcup\sqcup RSRS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSSS$
$s_\times$
$:0\sqcup\sqcup \to :0\sqcup\sqcup SSLS$
$0\sqcup 0\sqcup \to 0\sqcup 0\sqcup RSRS$
$:0\sqcup\sqcup \to :0\sqcup\sqcup SLSS$
$:\sqcup 0\sqcup \to :\sqcup 0\sqcup SSLS$
s_{19}
s_{22}
s_{23}
s_{26}
s_{28}
$:0\sqcup\sqcup \to :0\sqcup\sqcup SSLS$
$0\sqcup\sqcup\sqcup \to 0\sqcup\sqcup\sqcup RSSS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSSS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSSS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RRSS$
$0\sqcup\sqcup\sqcup$
$:\sqcup\sqcup\sqcup \to :\sqcup\sqcup\sqcup RSRS$
$:\sqcup\sqcup\sqcup \to :\sqcup\sqcup\sqcup RSSS$
$0\sqcup\sqcup\sqcup \to 0\sqcup\sqcup\sqcup RSSS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSSS$
s_{20}
s_{21}
s_{24}
s_{25}
s_{29}
$:\sqcup\sqcup\sqcup \to :\sqcup\sqcup\sqcup SSLS$
$:\sqcup 0\sqcup \to :\sqcup 0\sqcup SSLS$
$0\sqcup\sqcup\sqcup \to 0\sqcup\sqcup\sqcup RSSS$
$00\sqcup\sqcup \to 00\sqcup\sqcup RRSS$
$:\sqcup 0\sqcup \to :\sqcup 0\sqcup SSLS$
$0\sqcup 0\sqcup \to 0\sqcup 0\sqcup RSRS$
s_{34}
s_{33}
s_{32}
s_{31}
s_{30}
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RLSS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSSS$
$+0\sqcup\sqcup \to +0\sqcup\sqcup RRSS$
$+0\sqcup\sqcup \to +0\sqcup\sqcup RRSS$
$+\sqcup 0\sqcup \to +\sqcup\sqcup\sqcup SSRS$
$00\sqcup\sqcup \to 00\sqcup\sqcup RLSS$
$00\sqcup\sqcup \to 00\sqcup\sqcup RRSS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup SSLS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSSS$
s_{35}
s_{40}
s_{41}
s_{44}
s_{45}
$\sqcup\sqcup\sqcup\sqcup \to \sqcup\sqcup\sqcup\sqcup RSSS$
$0\sqcup\sqcup\sqcup \to 0\sqcup\sqcup\sqcup RSSS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup SSLS$
$\sqcup\sqcup\sqcup\sqcup \to \sqcup\sqcup\sqcup\sqcup LSSS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RRSR$
$0\sqcup\sqcup\sqcup \to 0\sqcup\sqcup\sqcup LSSS$
$:\sqcup\sqcup\sqcup \to :\sqcup\sqcup\sqcup LSSS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup LSSS$
$+\sqcup 0\sqcup \to +\sqcup\sqcup\sqcup SSRS$
$+\sqcup 0\sqcup \to +\sqcup 0\sqcup SSLS$
$00\sqcup\sqcup \to 00\sqcup 0RRSR$
$0\sqcup\sqcup\sqcup \to 00\sqcup 0RRSR$
$:0\sqcup\sqcup \to :: \sqcup : RRSR$
$:\sqcup\sqcup\sqcup \to :: \sqcup : RRSR$
$\sqcup\sqcup\sqcup\sqcup \to \sqcup\sqcup\sqcup\sqcup SLSL$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSSS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSRS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup SSLS$
s_{36}
s_{39}
s_{42}
s_{43}
s_{46}
$\sqcup\sqcup\sqcup\sqcup \to \sqcup\sqcup\sqcup\sqcup SLSS$
$\sqcup 0\sqcup 0 \to \sqcup 0\sqcup 0SLSL$
$\sqcup:\sqcup:\to \sqcup:\sqcup:SLSL$
$0\sqcup 0\sqcup \to 0\sqcup 0\sqcup RSRS$
$0\sqcup\sqcup\sqcup \to 0\sqcup 0\sqcup RSRS$
$+\sqcup 0\sqcup \to +\sqcup 0\sqcup SSLS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup SSLS$
$\sqcup 0\sqcup\sqcup \to \sqcup\sqcup\sqcup\sqcup SRSS$
$\sqcup 0\sqcup\sqcup \to \sqcup\sqcup\sqcup\sqcup SSSL$
$+\sqcup 0\sqcup \to +\sqcup 0\sqcup SSLS$
s_{37}
s_{38}
s_{51}
s_{50}
s_{47}
$+0\sqcup\sqcup$
$s_\times$
$\sqcup\sqcup\sqcup\sqcup \to \sqcup\sqcup\sqcup\sqcup SLSS$
$0\sqcup +\sqcup \to 0\sqcup +\sqcup LSSS$
$+\sqcup\sqcup\sqcup \to +\sqcup +\sqcup LSSS$
$+\sqcup\sqcup\sqcup \to$
$+\sqcup\sqcup\sqcup SSLS$
$\sqcup 0\sqcup\sqcup \to \sqcup\sqcup\sqcup\sqcup SRSS$
$0\sqcup 0\sqcup \to 0\sqcup 0\sqcup RSRS$
$0\sqcup\sqcup\sqcup \to 0\sqcup 0\sqcup RSRS$
$+00\sqcup \to +00\sqcup SRRS$
$+\sqcup +\sqcup \to +\sqcup +\sqcup LSSS$
$+:0\sqcup \to +:0\sqcup SSLS$
$+:\sqcup\sqcup \to +:\sqcup\sqcup SSLS$
$+:\sqcup\sqcup \to :+\sqcup\sqcup SRRS$
s_{54}
s_{53}
s_{52}
$+\sqcup +\sqcup \to +\sqcup +\sqcup LSSS$
$+\sqcup +\sqcup \to +\sqcup +\sqcup LSSS$
$+\sqcup\sqcup\sqcup \to +\sqcup\sqcup\sqcup RSRS$
$+\sqcup +\sqcup \to +\sqcup +\sqcup LSRS$
s_{49}
$+\sqcup 0\sqcup \to +\sqcup 0\sqcup SSLS$
s_{48}
s_{55}
$\sqcup\sqcup\sqcup\sqcup \to \sqcup\sqcup\sqcup\sqcup SSLS$
$0\sqcup +\sqcup \to 0\sqcup +\sqcup LSSS$
$0\sqcup\sqcup\sqcup \to 0\sqcup 0+LSRS$


s_{56}
$\sqcup\sqcup+\sqcup\to\sqcup\sqcup+\sqcup\,RSSS$
s_{57}
$+\sqcup+\sqcup\to+\sqcup+\sqcup\,RSSS$
s_{58}
$+\sqcup+\sqcup\to+\sqcup+\sqcup\,RSSS$
s_{59}
$+\sqcup+\sqcup\to+\sqcup+\sqcup\,RSSS$
s_{60}
$\sqcup\sqcup0\sqcup\to\sqcup\sqcup0\sqcup\,SSLS$
$0\sqcup+\sqcup\to0\sqcup+\sqcup\,RSSS$
$0\sqcup+\sqcup\to0\sqcup+\sqcup\,RSSS$
$+\sqcup+\sqcup\to+\sqcup+\sqcup\,RSSS$
s_{65}
$+\sqcup0\sqcup\to+\sqcup:\sqcup SSLS$
s_{64}
$+\sqcup+\sqcup\to+\sqcup+\sqcup\,SSRS$
s_{63}
$+\sqcup+\sqcup\to+\sqcup+\sqcup\,RSSS$
s_{62}
$+\sqcup+\sqcup\to+\sqcup+\sqcup\,RSSS$
s_{61}
$+\sqcup+\sqcup\to+\sqcup+\sqcup\,SSRS$
$0\sqcup+\sqcup\to0\sqcup+\sqcup\,RSSS$
s_{66}
$+\sqcup0\sqcup\to+\sqcup0\sqcup\,SSRS$
s_{67}
$+\sqcup0\sqcup\to+\sqcup0\sqcup\,SSLS$
s_{68}
$+\sqcup0\sqcup\to+\sqcup0\sqcup\,SSLS$
s_{69}
$+\sqcup+\sqcup\to+\sqcup+\sqcup\,RSSS$
s_{70}
$+\sqcup\sqcup\sqcup$
$+\sqcup\sqcup\sqcup$
$s_\times$
$s_\star$
$+\sqcup:\sqcup\to+\sqcup:\sqcup SSRS$
$+\sqcup:\sqcup+\sqcup:\sqcup SSLS$
$+\sqcup+\sqcup\to+\sqcup+\sqcup\,RSRS$
$0\sqcup+\sqcup\to0\sqcup+\sqcup\,RSSS$
$:\sqcup+\sqcup\to:\sqcup+\sqcup RSSS$
$0:+:\to0:+:SLSL$
$:0+0\to:0+0SLSL$
s_{77}
$0\sqcup+\sqcup\to0\sqcup+\sqcup\,RSSS$
$:\sqcup+\sqcup\to:\sqcup+\sqcup RSSS$
s_{76}
s_{75}
$:\sqcup+\sqcup\to:\sqcup+\sqcup RRSR$
s_{74}
$:\sqcup0\sqcup\to:\sqcup0\sqcup SSLS$
s_{71}
$0\sqcup:\sqcup\to0\sqcup:\sqcup RSRS$
$00+0\to00+0RRSR$
$:\sqcup:\sqcup\to:\sqcup:\sqcup SSLS$
$00+0\to00+0SLSL$
$:0+0\to:0+0SLSL$
s_{83}
$::+:\to::+:RSRS$
$+\sqcup+\sqcup\to+\sqcup+\sqcup\,RSSS$
$0\sqcup0\sqcup\to0\sqcup0\sqcup\,SSLS$
$:\sqcup:\sqcup\to:\sqcup:\sqcup SSLS$
s_{78}
$+\sqcup+\sqcup\to+\sqcup+\sqcup\,RSSS$
s_{79}
$::0:\to::0:SSLS$
s_{80}
$+\sqcup+\sqcup\to+\sqcup+\sqcup\,RSSS$
s_{73}
$0\sqcup+\sqcup\to0\sqcup+\sqcup\,RSSS$
$:\sqcup+\sqcup\to:\sqcup+\sqcup RSSS$
s_{72}
$0\sqcup+\sqcup\to0\sqcup+\sqcup\,LSSS$
$+\sqcup+\sqcup$
$s_\times$
s_{70}
$0:::\to0:::RSRS$
$0:0:\to0:::RSRS$
s_{70}
$::::\to::0:SSRS$
$0\sqcup+\sqcup\to0\sqcup+\sqcup\,RSSS$
$:\sqcup+\sqcup\to:\sqcup+\sqcup RSSS$
$0\sqcup:\sqcup\to0\sqcup:\sqcup SSLS$
$:\sqcup:\sqcup\to:\sqcup:\sqcup SSLS$
s_{83}
$::::\to::::SSLS$
s_{82}
$::\sqcup:\to::\sqcup:SSLS$
s_{81}
$::::\to::::SSLS$
$::0:\to::0:SSLS$
$:\sqcup+\sqcup\to:\sqcup+\sqcup SLSS$
s_{88}
$:0+\sqcup\to:0+\sqcup SLSS$
s_{89}
$:0+\sqcup\to:0+\sqcup SLSS$
$::+\sqcup\to::+\sqcup SSLS$
$:\sqcup+\sqcup\to:\sqcup+\sqcup SLSS$
$::+:\to::+:SLSS$
s_{89}
$::::\to::0:SSRS$
$:\sqcup+\sqcup\to:\sqcup+\sqcup SLSS$
$:0+:\to:0+:SSSR$
$::+:\to::+:SSSR$
$:\sqcup+:\to:\sqcup+:SSSR$
$:0+\sqcup\to:\sqcup+\sqcup SRSS$
$::+\sqcup\to:\sqcup+\sqcup SRSS$
$:\sqcup+\sqcup\to:\sqcup+\sqcup SLSS$
s_{84}
$:\sqcup+:\to:\sqcup+:RRSS$
s_{85}
s_{86}
s_{87}
$:0+\sqcup\to:0+\sqcup SSLS$
$::+\sqcup\to::+\sqcup SSLS$
s_{90}
$00+:\to00+:RRSS$
$0:+:\to00+:RRSS$
$0\sqcup+:\to00+:RRSS$
$:0+:\to:0+:SLSS$
$:0+0\to:0+0SRSR$
$::+0\to::0+0SRSR$
$:\sqcup+0\to:0+0SRSR$
$:0+:\to::+:SRSR$
$::+:\to::+:SRSR$
$:\sqcup+:\to::+:SRSR$
$:0+\sqcup\to:0+\sqcup SRSS$
$::+\sqcup\to:\sqcup+\sqcup SRSSS$
$:\sqcup+\sqcup\to:\sqcup+\sqcup SSSL$
$:\sqcup+0\to:\sqcup+\sqcup SSSR$
$:\sqcup+:\to:\sqcup+\sqcup SSSR$
s_{95}
$:\sqcup+0\to:\sqcup+\sqcup SSSR$
$:\sqcup+:\to:\sqcup+\sqcup SSSR$
$:\sqcup+0\to:\sqcup+\sqcup SRSR$
$:\sqcup+:\to:\sqcup+\sqcup SRSR$
$:\sqcup+\sqcup\to:\sqcup+\sqcup SRSR$
s_{94}
s_{93}
$:\sqcup+\sqcup\to:\sqcup+\sqcup SRSR$
s_{92}
$:\sqcup+\sqcup\to:\sqcup+\sqcup SSSL$
s_{91}
$:0+0\to:0+0SRSR$
$::+0\to::+:SRSR$
$:0+:\to::+:SRSR$
$:0+0\to:0+0SRSR$
$::+:\to::+:SRSR$
$:0+\sqcup\to:0+0SRSR$
$::+0\to::+:SRSR$
$:0+:\to:0+:SRSR$
$:0+\sqcup\to:0+0SRSR$
$::+\sqcup\to::+:SRSR$
$:\sqcup+0\to:\sqcup+0SSSL$
$:\sqcup+:\to:\sqcup+:SSSL$
$:\sqcup+0\to:\sqcup+0SSSL$
$:\sqcup+:\to:\sqcup+:SSSL$
$:\sqcup+\sqcup\to:\sqcup+\sqcup SSSL$
$:\sqcup+\sqcup\to:\sqcup+\sqcup SLSL$
$:\sqcup+\sqcup\to:\sqcup+\sqcup RSSS$
$0:+:\to0:+:RSSS$
$0\sqcup+\sqcup\to0\sqcup+\sqcup\,RSSS$
s_{96}
$:\sqcup+0\to:\sqcup+0SLSS$
s_{97}
$:\sqcup+\sqcup\to:\sqcup+\sqcup SLSL$
s_{98}
$:0+0\to:0+0SRSR$
s_{99}
$:\sqcup+\sqcup\to::+:RSSS$
s_{100}
$:0+0\to:0+0SLSL$
$::+:\to::+:SLSL$
$:\sqcup+\sqcup\to:\sqcup+\sqcup SSSL$



The formal description of U is $U = \{\{s_0, ..., s_{110}, s_\star, s_\times\}, \{0, 1 :\}, \{0, 1, :, \sqcup\}, \delta, s_0, s_\star, s_\times\}$, where the transition function can be read off the state diagram. ■

Definition 4.10 : Let $A = \{\langle T, w\rangle \mid T \text{ is a Turing machine and } T \text{ accepts } w\}$.

Theorem 4.11 : A is Turing-recognizable.

Proof : Let U' have a universal Turing machine, such as in Theorem 4.9, as a sub-Turing machine. U simulates T on w and U' returns accept if T accept w and reject if T rejects w. This Turing machine U' certainly recognizes A. ■

Remark : Note that if T loops on w in the above construction of U', then U' would also loop. Hence U' is not a decider of the language A.

Theorem 4.12 : A is undecidable.

Proof : Let $\langle T\rangle$ be the encoding of T. Suppose that such a decider D of A exists. Now construct a new Turing machine Q that has D as a submachine. One will feed to this machine the encoding $\langle T\rangle$ of a Turing machine T. Q will use the decider D on the input $\langle T, \langle T\rangle\rangle$. That is D will determine where T accepts its own encoding. If D accepts, then Q rejects $\langle T\rangle$. If D rejects , then Q accepts. Note that since D is a decider, Q will aways return accept or reject, that is it is a decider. Clearly, Q is itself a Turing machine. What happens when Q is given its own encoding as input. If Q accepts $\langle Q\rangle$, then D returns accept, so Q returns the opposite, that is Q reject $\langle Q\rangle$. This case is impossible. If Q rejects $\langle Q\rangle$, then D returns reject, so Q returns the opposite, that is Q accepts $\langle Q\rangle$. This is also impossible. However, Q is a decider because D is a decider, but neither accept nor reject is possible for Q on $\langle Q\rangle$. This is a contradiction. ■

Remarks : One says that A is reducible to another problem B if a solution to B would give a solution to A. Furthermore, if A is reducible to B and there is no solution to A, then there must be no solution to B. This method of reducibility allows one to find other undecidable problems from a known undecidable problem like A above.

Theorem 4.13 : (Halting Problem) Let $H = \{\langle T, w\rangle \mid T \text{ is a Turing machine that halts on input } w\}$. H is

undecidable.

Proof : Suppose H is decidable with the decider Turing machine D_H. D_H accepts $\langle T, w\rangle$ if T accepts or rejects w, but rejects $\langle T, w\rangle$ if T loops on w. Using D_H, one shall create a decider Turing machine D_A which will decide A. D_A will use D_H and a Universal Turing machine, like in Theorem 4.9, to simulate what T does on w. D_A will function as follows: given an arbitrary Turing machine T and an input w, give its encoding $\langle T, w\rangle$ to D_H. If D_H rejects $\langle T, w\rangle$, then D_A rejects it also. If D_H accepts, then run U on $\langle T, w\rangle$. If U accepts $\langle T, w\rangle$ (which means T accepts w), then D_A accepts. If U rejects (which means T rejects w), then D_A rejects. U must either reject or accept $\langle T, w\rangle$ since D_H has already determined that T will halt on w. This D_A would be a decider of A. Contradiction! It was already shown above that A is undecidable. ■

Theorem 4.14 : Let $E = \{\langle T\rangle \mid T \text{ is a Turing machine and } L(T) = \emptyset\}$. E is undecidable.

Proof : Suppose that E can be decided by a Turing machine D_E. One wishes to construct a decider Turing machine D_A for A. So suppose T is a Turing machine, and w is some string. From the description or encoding $\langle T, w\rangle$, D_A constructs the encoding of a new Turing machine T_w which behaves as follows. If x is a string such that $x \neq w$, then T_w rejects. If $x = w$, then run T on w and accept w if T accepts w. Clearly, constructing and running T_w from $\langle T, w\rangle$ is an algorithmic procedure. The checking of whether $x = w$ can easily be accomplish if w is built into the description of T_w, and T_w would merely check symbol by symbol whether $x = w$. After T_w has been constructed, D_A will run D_E, the decider of E, on T_w. If D_E accepts $\langle T_w\rangle$, then D_A rejects $\langle T, w\rangle$, because D_E accepting T_w means that $L(T_w)$ is empty. If D_E rejects $\langle T_w\rangle$, then accept. By how T_w was constructed, the only possible string that T_w could accept was w. If the language is empty, then T_w must reject or loop on w, which only happens if T rejects or loops on w. Similarly, D_E rejecting T_w means that $L(T_w) \neq \emptyset$. By constructing T_w can only possibly accept w. So if the language is nonempty, the language contains only one string, i.e. w. If T_w accepts w, then T accepts w. This D_A is a decider for A. Contradiction! ■

Theorem 4.15 : Let $Q = \{\langle F_1, F_2\rangle \mid F_1 \text{ and } F_2 \text{ are Turing machines such that } L(F_1) = L(F_2)\}$. Q is undecidable.

Proof : This is a proof by reducing E, above, to Q. Suppose Q was decidable by a Turing machine D_Q. Find or construct a Turing machine R that rejects all input, that is $L(R) = \emptyset$. For instance the Turing machine that goes from s_0 to $s_\times$ on every input symbol is such a Turing machine. Now one wishes to construct a Turing machine D_E using D_Q. The construction is as follows. For any Turing machine T, run D_Q on $\langle T, R\rangle$. If D_Q accepts, then D_E accepts because T has the same language as R, i.e. $L(T) = L(R) = \emptyset$. If D_Q rejects, then D_E rejects because $L(T) \neq L(R) = \emptyset$. D_E would be a decider for E, but this contradicts Theorem 4.14. ■

Theorem 4.16 : Let $R = \{\langle T\rangle \mid T \text{ is a Turing machine and } L(T) \text{ is a regular language.}\}$. R is undecidable.

Proof : Suppose that R is decidable by a Turing machine D_R. One will seek to create a decider D_A that decides the acceptance problem A. First, recall from Theorem 2.17, the set of all strings over an alphabet Λ, that is Λ^*, is a regular language. By Theorem 2.26, the language $\{0^{n^2} \mid n \in \mathbb{N}\}$ is a nonregular language.

Now construct D_A as follows. Given an arbitrary Turing machine T and string w, from the encoding $\langle T, w\rangle$, construct the Turing machine T_w with an nonempty alphabet Λ. Let $a \in \Lambda$. T_w functions as follows: For some x a string over Λ, (it is helpful to think of $x \in \Lambda^*$), if x is in the form a^{n^2} for some $n \in \mathbb{N}$, then T_w accepts x. If x is not in this form, then T_w accepts x if T accepts w. Now run D_R on $\langle T_w\rangle$. If D_R accepts,

then D_A accepts. If D_R rejects, then D_A rejects. D_A is a decider for A.

To see this, note that from the construction of T_w, one sees that T_w accepts the set of every string (i.e. Λ^*), a regular language, if T accepts w. If T does not accept w, then T_w will only accept strings of the form a^{n^2} which is a nonregular language. That is if T accepts w, then $L(T_w)$ is a regular language. If T does not accept w, then $L(T) = \{a^{n^2} \mid n \in \mathbb{N}\}$ is a nonregular language. If D_R decides if a Turing machine has a regular language or not, then D_R can tell whether T_w accepts a regular or non-regular language. Therefore, D_A constructed using D_R can decided A. Since one knows that A is undecidable, this leads to a contradiction. ■

Theorem 4.17 : (Rice's Theorem) If P is a language consisting of Turing machine encodings such that
(1) P is nontrivial - P is not empty and does not contain every Turing machine.
(2) If T_1 and T_2 are Turing machines and $L(T_1) = L(T_2)$, then $\langle T_1 \rangle \in P$ if and only if $\langle T_2 \rangle \in P$.
Then P is an undecidable langauge.

Proof : This is proved by reducing A to P. Let D_P be a decider of P. Let $\langle T_\emptyset \rangle$ be an encoding of a Turing machine that rejects all strings; such a Turing machine exists. One can assume that $\langle T_\emptyset \rangle \notin P$, for otherwise one could work with P^c. Let T be a Turing machine such that $\langle T \rangle \in P$. Note that $L(T) \neq \emptyset$, since if $L(T) = \emptyset = L(T_\emptyset)$, by condition (2), $\langle T \rangle \notin P$ because by assumption $\langle T_\emptyset \rangle \notin P$. Also note that condition (1) shows that there exists a $\langle T \rangle$ that is not in the same set (P or P^c) as $\langle T_\emptyset \rangle$.

D_A works as follows: First, from the encoding $\langle M, w \rangle$ where M is an arbitrary Turing machine, D_A constructs the Turing machine M_w. M_w behaves as follows. Given an input string x, if M rejects w, then M_w rejects. If M accepts w, then simulate T on x. If T accepts x, then M_w accepts. If T reject x, then M_w rejects x.

Now run D_P on $\langle M_w \rangle$. If D_P accepts, then D_A accepts. If D_P rejects $\langle M_w \rangle$, then D_A rejects. This D_A is a decider for A.

Note that if M accepts w, then $L(M_w) = L(T)$. One knows that $\langle T \rangle \in P$, so by condition (2), $\langle M_w \rangle \in P$. If M rejects w or M loops on w, then $L(M_w) = L(T_\emptyset) = \emptyset$ and using the fact that $T_\emptyset \notin P$, so by condition (2), $\langle M_w \rangle \notin P$. One reduces A to deciding whether $\langle M_w \rangle \in P$. Since it is known that A is undecidable, P is also undecidable. ■

Theorem 4.18 : Let $B = \{\langle T \rangle \mid T$ is a Turing machine that halts on ϵ, in other words, $\epsilon \in L(T)\}$. B is undecidable.

Proof : B is the set of all encoding of Turing machine that halts on the tape of just blank symbols. Certainly B is nontrivial. Let $\sqcup$ represent the blank symbol, s_0 be the start state, $s_\star$ as the accept state, and $s_\times$ be the reject state. A Turing machine with $(s_0, \sqcup, s_0, \sqcup, R) \in \delta$ will loop on the tape of all blank symbols. This Turing machine is not in B. Certainly there are Turing machines that do halt on the empty string. An example is any Turing machine that has $(s_0, \sqcup, s_\star, \sqcup, R) \in \delta$. Also any Turing machine that has $(s_0, \sqcup, s_\times, \sqcup, R) \in \delta$ will reject on the empty string. Therefore, B is not an empty language and not the language of all Turing machine encodings.

Now suppose T_1 and T_2 are Turing machines such that $L(T_1) = L(T_2)$. If $\langle T_1 \rangle \in B$, then $\epsilon \in L(T_1) = L(T_2)$, so $\epsilon \in L(T_2)$ so $\langle T_2 \rangle \in B$. Similarly, if $\langle T_2 \rangle \in B$, then $\epsilon \in L(T_2) = L(T_1)$, which implies that $\langle T_1 \rangle \in B$. By Rice's Theorem, B is undecidable. ■

Remarks: If one actually wanted to prove the undecidability of B by reducing some undecidable problem to B and constructing an actual decider, one could do so by simply imitating the proof of Rice's Theorem with B is place of the arbitrary language P.

Section 5: Computable Functions

Remarks : In this section, the concept of reducibility will be made more formal through comptuable functions and mapping reducibility.

Definition 5.1 : Let Λ be an alphabet. A function $f : \Lambda^* \to \Lambda^*$ is a (Turing) Computable Function if some Turing machine T, on input w over Λ^*, halts with $f(w)$, as the nonblank symbols, on its tape.

Definition 5.2 : Let Λ be an alphabet. A language A, over Λ, is Mapping Reducible to language B, over Λ, denoted $A \leq_m B$, if there is a computable function $f : \Lambda^* \to \Lambda^*$ such that for every string w, $w \in A$ if and only if $f(w) \in B$. One calls the function the reduction of A to B.

Theorem 5.3 : If $A \leq_m B$ and B is decidable, then A is decidable.

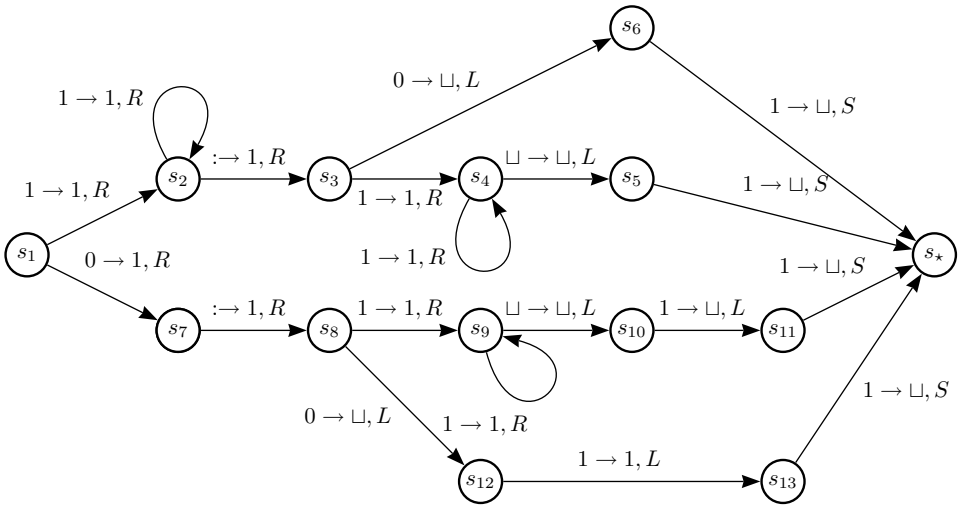
Proof : Assume that A and B is over the same alphabet or else encode every string of one language into the other alphabet. Because B is decidable it has a decider Turing machine called D_B. Let T_f be the Turing machine that computes the value of the function f, the computing function (in the definition of mapping reducibility). One can construct a decider D_A for A as follows:

On any input w. D_A will first run T_f as a sub-Turing machine to compute the value of $f(w)$. Then D_A will run D_B on $f(w)$. If D_B accepts $f(w)$, then D_A accepts. If D_B rejects $f(w)$, then D_A rejects. Note that D_A accepts w if and only if D_B accepts $f(w)$. By definition of mapping reducibility, $w \in A$ if and only if $f(w) \in B$. Since D_B is a decider for B it accepts exactly those strings in B. Thus, D_A, as constructed, accepts exactly those things in A. So, D_A is the desired decider. ■

Theorem 5.4 : The function $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is a computable function.

Proof : This is the addition function of $\mathbb{N}$. Note that ordinarily the natural number are represented over the alphabet $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. For this paper, to create the computing Turing machine for the addition function $+$, one uses the alphabet $\{0, 1, :\}$. Let $\langle 0 \rangle = 0$. For $n \in \mathbb{N}$ such that $n > 0$, $\langle n \rangle = 1...1$, n times. Since the addition function takes an ordered pair as its argument, the : symbol is the delimiter. That is, for $a, b \in \mathbb{N}$, $\langle (a, b) \rangle = \langle a \rangle : \langle b \rangle$. So $\langle (3, 4) \rangle = 111 : 1111$ and $\langle (0, 6) \rangle = 0 : 111111$.

The state diagram of the computing Turing machine is given below:



What remains on the tape is the value $+(a,b)$. Recall that if 0 is the value of some function, then $\langle 0 \rangle = 0$ would be left. This Turing machine operates by merely combining the 1's, by removing the delimiter for :. However, one must be careful, when 0 is being added. The formal description of this Turing machine is $\{\{s_1, ..., s_{13}, s_\star\}, \{0, 1, :\}, \{0, 1, :, \sqcup\}, \delta, s_1, s_\star, s_\times\}$, where the δ can be read off the state diagram. On this machine, one only considers what is left on the tape when the machine accepts. One does not consider what is left if the machine rejects, since this means the string was not even a proper encoding. ■

Theorem 5.5 : Let Λ be some alphabet. $f : \Lambda^* \to \Lambda^*$ be a computable function, then $\{(w, f(w)) \mid w \in \Lambda^*\}$ is decidable.

Proof : Suppose that the ordered pair has an appropriate encoding. Suppose T is the computing Turing machine for f. One can create a decider D for this language. It would be a two tape Turing machine. First, it check whether the input string is an correct encoding of some ordered pair made of string in Λ^*. Then run T on w on the second tape. T must halt on w by definition of the computing Turing machine. Then check whether the second coordinate is the encoding of $f(w)$ which is left on the tape simulating T on w. D accepts the ordered pair if it is, and rejects if it does not match. ■

Remarks : For the next problem, the Turing machine is as defined in Definition 3.7. That is, it has a finite set of states called the halting states (no distinction between accept and reject) and moves left or right. Recall the transition function has $|\Sigma|\ |\Gamma|$ number of elements in its codomain. Each element of the codomain is mapped to an ordered 3-tuple, a state, symbol, and direction. Let $|\Sigma| = a$ and $|\Gamma| = b$, where $a, b > 0$. If $\mathbb{N}^\times = \mathbb{N} \backslash \{0\}$, then $a, b \in \mathbb{N}^\times$. Then there are $(2ab)^{ab}$ number of transition functions and hence that many Turing machines with a states and b symbols in the Tape alphabet. Running all of them on the blank tape (or string ϵ), some will halt and some with loop. The ones that halt are known as busy beavers. Of those Turing machine that halt on the blank tape, let c be the number of steps needed by the Turing machine, that takes the longest to halt, to halt. This value c is the value of the Busy Beaver Function f on (a, b).

Definition 5.6 : Let $a, b \in \mathbb{N}^\times$. For any Turing machine T, let Σ_T and Γ_T be the set of state and tape alphabet of T. Let $B_{a,b} = \{T \mid |\Sigma| = a,\ |\Gamma| = b,\ \text{and } T \text{ halts on } \epsilon\}$. Define $B = \{T \mid T \text{ is a Busy Beaver}\}$. Let $g : B \to \mathbb{N}$ be defined by $g(T) =$ number of steps it takes T to halt on ϵ. Then define the Busy Beaver Function $f : \mathbb{N}^\times \times \mathbb{N}^\times \to \mathbb{N}$ by

$$f(a, b) = \max\{g(T) \mid T \in B_{a,b}\}$$

Theorem 5.7 : The busy beaver function is not computable.

Proof : For this proof, suppose Λ is the language of the encoding of everything in consideration.

Suppose the Busy Beaver function f is computable. Then by Theorem 5.5, the set $X = \{(a, b, c) \mid a, b \in \mathbb{N}^\times \text{ and } c = f(a, b)\}$ is decidable. Of course this information must be encoded in Λ. Let $B = \{\langle T \rangle \mid T \text{ accepts } \epsilon\}$ be the undecidable language from Theorem 4.18.

Next to show $B \leq_m X$, (B is mapping reducible to X) one must create the desired computing function h. First, note that if f is computable, then there is a Turing machine that can check whether a string is an encoding of a Turing machine in B and, from the encoding, leave on its tape the cardinality of that machine's set of state or tape alphabet. For all input that are not an encoding of Turing machines in B, the Turing machine leaves one string which is not a legal or proper encoding of an ordered pair of natural numbers. Let w be some string. The Universal Turing machine constructed in the paper had the ability to check if w was a Turing machine encoding and read the number of elements in $a = \Sigma_w$ and $b = \Gamma_w$ and write the encodings of those numbers on one of its four tapes. If w is a Turing machine encoding, one can use a modified Universal Turing machine U' to check if it is in B. U' will simulate the encoded Turing machine w

while keeping record of the number of steps it has gone through. It accepts w if it halts on or before $f(a,b)$ number of steps (the Busy Beaver Function) and rejects w if on the $f(a,b)^{\text{th}}$ step, the simulation is not in an accept state. In fact, this U' is a decider of B, which one knows is undecidable, so the proof could end here; however, one wishes to use f to construct a mapping reduction between B and X.

From the above, it is clear that there is a computable function $g : \Lambda^* \to \Lambda^*$ such that $g(w) = \langle (a,b) \rangle$ when $w = \langle T \rangle$ is an encoding of a Turing machine in B and $a = \Sigma_T$ and $b = \Gamma_T$. If w is not a legal encoding of any Turing machine, map $g(w)$ to a string in Λ^* which is not a legal encoding of any (a,b) such that $a, b \in \mathbb{N}^\times$.

Next, supposing the busy beaver function is computable, one can create a Turing machine such that given an encoding of (a,b) such that $a, b \in \mathbb{N}^\times$, it returns the encoding of (a,b,c) such that $c = f(a,b)$, where f is the busy beaver function. This Turing machine would merely read a, b off its tape, run the computing Turing machine T_f on $\langle a,b \rangle$ and write the encoding of $c = f(a,b)$ at the end of the input (or some other reasonable manner according to the encoding). For all strings that are not encodings of ordered pairs of positive natural numbers, the Turing would leave a string on its tape which is not a legal encoding of a 3-tuple of natural numbers. Define the function p as follows, $p : \Lambda^* \to \Lambda^*$ such that when $w = (a,b)$, where $a, b \in \mathbb{N}^\times$, $p(w)$ is mapped to $\langle (a,b,c) \rangle$, where $c = \langle f(a,b) \rangle$ and when w is not a legal encoding of such an ordered pair of natural number, $p(w)$ is a string that is not a legal encoding of a 3-tuple of natural numbers. The above Turing machine shows that p is a computable function.

Recall computable functions have the same set as its domain and codomain. Clearly composition of computable functions are computable. Merely one run the computing Turing machine on the argument of the first function. What is left on the tape is the argument of the second function. The range of the first function is in the domain of the second function by definition, so everything is well defined. This shows that the function $h = p \circ g$ is a computable function.

One claims that this h is the function which makes $B \leq_m X$. So suppose that $\langle T \rangle \in B$. $h(\langle T \rangle) = p(g(\langle T \rangle)) = p(\langle (a,b) \rangle) = \langle (a,b,c) \rangle$, where $a = |\Sigma_T|$, $b = |\Gamma_T|$, $c = f(a,b)$. However, $h(T) = (a,b,c) \in X$ by definition of X. Suppose $h(w) = (a,b,c) \in X$. This means that $w = \langle T \rangle$ such that T is a Turing machine in B. To prove this, suppose that w is not an encoding of a Turing machine in B. By how the function p and g where defined on those strings which were not encodings of Turing machine in B and not ordered pairs of positive natural numbers, respectively, $h(w)$ could not be a encoding of a 3-tuple in X. Contradiction! Therefore $B \leq_m X$. By Theorem 5.5, X is decidable. By Theorem 5.3, B is decidable. However this contradicts Theorem 4.18, which asserted that B is undecidable. Therefore, the Busy Beaver function f can not be computable. ■

References

Rudin, Walter. *Principle of Mathematical Analysis*. 3rd ed. New York: McGraw-Hill, 1976. International Ser. in Pure and Applied Mathematics.

Sally, Paul J. *Tools of the Trade: an Introduction to Advance Mathematics*. Providence, Rhode Island: American Mathematical Society, 2008.

Sipser, Michael. *Introduction to the Theory of Computation*. 2nd ed. Boston: Thomson Learning, Inc., 2006.

Acknowledgement

I would like to thank my mentors Matthew Wright and Jonathan Stephenson whose help was crucial to the completion of this paper. I am grateful to them for introducing me to Computatability Theory, directing me to the appropriate resources, and guiding me through my study. Their careful reading and revision of this paper were invaluable in reaching the present quality of the Mathematics (and English!) in this paper. Any further errors are solely the fault of the author.