Metamathematics

Jacob Denson

January 24, 2016

Table Of Contents

Chapter 1

Introduction

Metamathematics is the formal analysis of the notion of deduction and proof in mathematics. The general objective is to mathematically model the procedure of proof, so that we can understand its limitations, and how certain paradoxes arise in the discussion of mathematical theory. The latter is the context in which the theory arose.

Example (Russell)**.** *Consider the following set of objects.*

$$X = \{x : x \notin x\}$$

Russell's paradox rests on the following question: Is $X \in X$? If this were true, then since X only contains sets which do not contain themselves, $X \notin X$, a contradiction. Conversely, if $X \notin X$, then, by definition, $X \in X$. Thus we are lead to believe that X cannot contain itself, nor can it contain itself! In more colloquial terms, consider a town with a single barber, who shaves everyone who does not shave themselves. Does the barber shave himself?

Example (Cantor)**.** *For any set X, Cantor proved that there is no injective function from $\mathcal{P}(X)$ to X. But if we take X to be the set of all things, then $\mathcal{P}(X) \subset X$, so surely there is an injective function from $\mathcal{P}(X)$ to X.*

Example (Burali-Forti)**.** *No ordinal is order-isomorphic to a proper segment of itself. The well-ordering principle tells us every set can be well-ordered, and thus associated (by an order-isomorphism) with an ordinal. But then the set of all ordinals, being a set, is order-isomorphic to an ordinal, which must be an initial segment of itself.*

Example (Richard)**.** *There are only countably many english expressions, yet uncountable real numbers. Consider a particular enumeration of all real numbers which can be described in the english language. Then, take the expression*

> *"The number constructed by Cantor's diagonal argument."*

This describes a finitely describable number which isn't in the enumeration.

Example (Löb)**.** *Consider the Preposition B, defined*

$$B \implies A$$

If B is true, then $B \implies A$ is true, so A is true. But then $B \implies A$ is true, so B is true, and we conclude A is true. But A was arbitrary, so we conclude that every logical statement is true!

Example (Curry)**.** *Löb's semantical argument has a syntactic equivalent, due to Haskell Curry. Consider*

$$C = \{x : (x \in x) \implies P(x)\}$$

Then $(C \in C)$ is true if and only if $(C \in C) \implies P(C)$. We must have $C \in C$, for if $C \notin C$, then $(C \in C) \implies P(C)$ holds vacantly, a contradiction. But this implies $(C \in C) \implies P(C)$, so $P(C)$ is true. We conclude $P(C)$ is true, irrespective of what $P(C)$ actually says.

To resolve these paradoxes, we must attempt to understand the process of mathematics in acute detail. What constitutes a proof of a statement? What does it mean for a statement to be 'True'? In fact, what standards do we use to define a statement[1]. As mathematicians, we possess a very precise weapon, above normal standards of philosophical rigour, which may reveal more then previous epistemology has achieved.

As the study of philosophy has shown (as an example, rather than the work of the field), it is rarely possible to completely define and agree on fundamental principles. All we can do is accept some properties of these principles, accepted by a portion of the populace, and logically derive consequences from them. The purpose is not to completely describe all knowledge of the principle, and it would be arrogant to attempt to, but instead to better understand the fundamental principle in question. When a

[1]are we to accept such a circular definition, as in Löb's paradox, or we must explain why the statement is unacceptable.

physicists models a physical system, the result is an ideal, separated from reality in all facts but the experimental parameters and observations. We will likely never create a model which exactly describes the evolution of the world (and it is myopic to suggest a model is perfect), but we can still do a pretty bloody good job!

The confusion that exists when understanding philosophy, and with understanding physics, is that once assumptions are stated one almost forgets the assumptions, and proceeds to derive results as if they were about the world, rather than a model. A physicist talks of "a planet moving according to the equation $\ddot{x} = -m/x^2$" even if he is actually talking about the dynamical system whose evolution is described by the differential equation $\ddot{x} = -m/x^2$, which *models* the motion of a planet. This is unavoidable, since thinking in terms of the physical systems favours intuition and promotes intellectual discovery. Keep this in mind as we begin a formal analysis of logic.

What we shall discuss is a model of mathematics. In the modelling process, some basic notions (such as intuition) may be mutilated; in pinning down some notions, other logical properties, hopefully extraneous, are lost. Nonetheless, we hope the theorems we shall prove give us correct insights into the use of mathematics and logic. For instance, Cohen's theorem shows that the so-called 'continuum hypothesis' cannot be proven with the conventional axioms of set theory. This gives evidence that there is no proof. If someone ends up proving the continuum hypothesis, this probably does not show a deep contradiction in mathematics, but merely that our formal model of logic doesn't model the right principles. All we can do in the present is hope that enough ethereal notions of logic are understood to avoid these problems. Whether mathematical logic has succeeded in this approach, is up to you to decide. No accepted proof of the continuum hypothesis has been discovered as of present.

There is a question we haven't answered which should hit right at the heart of our investigation:

How do we know a proof is true?

I don't mean the proofs in our formal models, but the proofs we will use to derive facts about our models. If we are proving statements about proving, we must take care in justifying our objective. When we see a deduction about Banach spaces, or semi-simple R-modules, how do we know a proof is correct. Most people would agree that a proof is an argument which starts with accepted assumptions, and proceeds by elementary and universally accepted deductions (accepted as 'obvious') to form the conclusion of the argument. But we can continue poking holes in this definition. What is an 'obvious' statement – to a functional analyst, the compactness of the character space of a Banach algebra is obvious, yet to a first year undergraduate the statement is complete nonsense. So we must accept that the correctness of a proof really is a relative concept. But then how we can prove concrete facts about subjective concepts? We could go on and on asking questions like this, and would end up with degrees in philosophy rather than mathematics, but the point is sound. We must argue whether a proof holds up to our own standards. Aside from an ultimate, experimental observation of the correctness of a statement in the real world (or a mental contradiction in the abstract world of mathematics), we can only rely on intuition to accept a proof.

It may seem circular to analyze mathematics through mathematics, but this is the whole reason for the use of the 'mathematical model of logic'. By placing our understanding mathematics in a model, we may treat the model as a physical object, avoiding the problems of circular definitions by creating a second definition modelling the original. This method shall completely encompass our adventure in mathematical logic, which we shall now begin.

Chapter 2

Prepositional Logic

We shall begin with the simplest model of truth – propositional logic. To understand mathematics, we model it in a mathematical system (known as a **formal language**), and then analyze the language using common notions of mathematics (known as the **metalanguage**). The standard model is analyzes strings – sequences of abstract symbols from a given **alphabet**. Each string represents a mathematical statement, and manipulating these strings models how a mathematician infers some mathematical statement from another. It is difficult to describe intuitively the calculus of strings works, so it is best to proceed swiftly, and see the tool work in action.

2.1 Syntax

Each abstract symbol in our alphabet shall represent a precise form of colloquial speech. Prepositional logic, which we shall begin with, analyzes basic notions of truth and falsity in logical statements. We treat some statements as **atomic**, both because of how basic the statements are and because our language will not have the capability to express the complete capability of the english language (it is just a model of logic, as we have already described). For instance, atomic statements shall represent claims like 'Socrates is a man', and 'every woman is a human', but not statements such as 'Socrates is a man and every woman is a human', for this statement can be broken into more basic terms in our system. In English, the fact that 'every woman is a human' can be broken into infinitely many statements, such as 'Julie is a human', 'Laura is a human', and so on, but we will be unable to model this type of statement in our current calculus. Predicate logic, which we shall discuss later, attempts to analyze a system which can express these types of sentences.

Definition. Let Λ be a set. Then the set Λ^* of **strings over** Λ, are all finite (and possibly empty) sequences with elements in Λ. The empty string () shall be denoted ε. A string $(v_1, \dots, v_n)$ is often denoted $v_1 v_2 \dots v_n$. We shall identify an element in Λ with the corresponding one letter string in Λ^*. The concatenation of two strings $s = s_1 \dots s_n$ and $w = w_1 \dots w_m$, denoted sw, is the string $s_1 \dots s_n w_1 \dots w_m$. A **substring** of a string is a consecutive subsequence of symbols.

For instance, $\{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 11, \dots\}$. If we view concatenation as an associative algebraic operation, then Λ^* is the smallest *monoid* containing Λ, relative to the operation of concatenation.

Definition. Let Λ be a set, which we shall often assume to be at least countable, and for technical reasons is disjoint from $\{(,), \wedge, \vee, \neg, \Rightarrow, \Leftrightarrow \}$. The **propositional language with atoms in** Λ, denoted $\text{SL}(\Lambda)$, is the smallest subset of $(\Lambda \cup \{(,), \wedge, \vee, \neg, \Rightarrow, \Leftrightarrow\})^*$ such that

1. $\Lambda \subset \text{SL}(\Lambda)$.

2. If $s, w \in \text{SL}(\Lambda)$, then $(\neg s), (s \wedge w), (s \vee w), (s \Rightarrow w), (s \Leftrightarrow w) \in \text{SL}(\Lambda)$.

An element of $\text{SL}(\Lambda)$ is called a **formula** or **statement**.

Each **connective** represents a certain linguistical way we combine statements in colloquial language, described below.

Connective	Name of Connective	Meaning of statement
$\neg s$	Negation	"s is *not* true"
$s \wedge w$	Conjuction	"s *and* w is true"
$s \vee w$	Disjunction	"s *or* w is true"
$s \Rightarrow w$	Implication	"*If* s is true, *then* w is true"
$s \Leftrightarrow w$	Bicondition	"s is true, *if, and only if,* w is true"

The way in which these conjunctive symbols represent the linguistic form will become clear later. For now, we shall treat them as abstract symbols with no intrinsic meaning.

Take care to notice that $\text{SL}(\Lambda)$ is the smallest such set satisfying these rules. This is defined in the same way that most 'smallest things' exist in mathematics (like a group generated from a set, or a subspace generated from a spanning set), because the intersection of sets satisfying these operations also satisfy these operations. From the definition, we obtain the incredibly important method of constructing proofs.

Theorem 2.1 (Structural Induction)**.** *Suppose we have a property we would like to prove about all elements of $SL(\Lambda)$. Suppose that the property is true of all elements of Λ, and that if the proposition is true of s and w, then the proposition is also true of $(\neg s), (s \wedge w), (s \vee w), (s \Rightarrow w)$, and $(s \Leftrightarrow w)$. Then the proposition is true for all of $SL(\Lambda)$.*

Proof. Note that the set

$$K = \{s \in (\Lambda \cup \{(,), \wedge, \vee, \neg, \Rightarrow, \Leftrightarrow\})^* : P \text{ is true of } s\}$$

satisfies axioms (1) and (2) of the definition of $\text{SL}(\Lambda)$, thus $\text{SL}(\Lambda) \subset K$, so that if $s \in \text{SL}(\Lambda)$, P is true of s. □

We will apply structural induction in a dense subset of the theorems of these notes. The next proof is just the beginning.

Theorem 2.2. *Any sentence in $SL(\Lambda)$ contains as many left as right brackets.*

Proof. Any atom in Λ contains no left brackets, and no right brackets, and thus the same number of each. Let s have n pairs of brackets, and let w have m. Then $(\neg s)$ contains $n+1$ pairs of brackets, and $(s \circ w)$ (where $\circ \in \{\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow\}$) contains $n+m+1$ brackets. By structural induction, we have proved our claim. □

This theorem is just a warmup. We need a more in depth theorem, to avoid issues with parsing statements. These issues are the reasons for brackets, just as in arithmetic, where $2+7-5-4$ is ambiguous (is it $9-1=8$, or $9-9=0$). There is a split in mathematical logic between syntax (studying strings), and semantics (understanding) strings. We are studying syntax to begin with, so that we may begin semantics (so we may calculate, as we have done in arithmetic).

Theorem 2.3. *If $wu = s \in SL(\Lambda)$, where $w \neq \varepsilon$, then w has at least as many left brackets as right brackets, and $w = s$ if and only if w has the same number of left and right brackets.*

Proof. If $s \in \Lambda$, then s is only one letter long, so $s = w$, and has no brackets. Now let s and s' satisfy the theorem. We split our proof into two cases.

1. $wu = (\neg s)$: Let us march through all cases (we leave it to the reader to check that w shall always have more left brackets than right). Suppose w has the same number of left brackets than right. w cannot equal (or $(\neg$, since they contain more left brackets than right brackets. Furthermore, w cannot equal $(\neg v$, where v is a prefix of s, because, by induction, v has at least as many left brackets as right brackets, and then w has more left brackets than right brackets. Thus w must equal $(\neg s)$.

2. $wu = (s \circ w)$, where $\circ$ is an element of $\{\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow\}$: Again, march through the letters of the string. w obviously cannot equal (, nor $(v$, where v is a substring of s, by induction. Similarily, w cannot equal $(s\circ$, nor $(s \circ v$, where v is a substring of w, so w must equal $(s \circ w)$.

And we have verified the statement by structural induction. $\square$

The importance of the next corollary allows us to make a recursivs structural definition. As an example, let $\Lambda = \{A, B, C\}$, and $\Gamma = \{X, Y, Z\}$. We would like to consider SL(Λ) and SL(Γ) the same by mapping formulas of one set to the other, extending the map $f : X \mapsto A$, $Y \mapsto B$, $Z \mapsto C$ to general formulas by

$$f((s \circ w)) = (f(s) \circ f(w)) \qquad f((\neg s)) = (\neg f(s))$$

But how do we know this map is well defined?

Corollary 2.4. *Every string in $SL(\Lambda)$ can be written in a unique way as either an element of Λ, or $(\neg s)$ and $(s \circ w)$, where s and w are elements of $SL(\Lambda)$.*

Proof. That such a representation exists should be trivial to show by now. Suppose we have two representations. If one of the representations is an element of Λ, the other representation must have a string of length one, and is therefore an element of Λ, and equal to the other. If we have two representations $(\neg s) = (\neg w)$, then by chopping off symbols, $s = w$, and thus the overall string is equal. It is impossible to have two distinct representations $(s \circ w) = (\neg u)$, for no element of $SL(\Lambda)$ begins with $\neg$, which s would have to satisfy. Finally, suppose we have two representations $(s \circ w) = (u \circ v)$. Then either u is a substring of s, or s is a substring of u, and one is the prefix of the other. But both have balanced brackets, which implies $s = u$, and by chopping letters away, $w = v$. Thus we have shown the representation is unique. □

Before we move on to semantics. It should be noted that our current discussion applies to any syntactical system formed by variables, 1-ary functions (like $\neg$), and 2-ary functions (like $\wedge$ and $\vee$). Thus when we discuss predicate logic, we shall be able to extend our proofs with some work to include the 1-ary functions "$\forall x :$" and "$\exists x :$", in addition to n-ary functions $f(x_1, \dots, x_n)$.

And before we leave the discussion of syntax, it is interesting to discuss a less natural, but syntactically simpler way of forming sentences, known as **polish notation**, after its inventor Jan Łukasiewicz. Rather than writing connectives in **infix notation**, like $(u \wedge v)$ and $(u \Rightarrow v)$, we write in **prefix notation**, like $\wedge uv$ and $\Rightarrow uv$. We do not need brackets to parse statements anymore, as one can prove, but it is much more easy to read the statement "$((a \Rightarrow (b \vee c) \wedge d)$", than "$\wedge \Rightarrow a \vee bcd$".

2.2 Semantics

We can understand the discussion above without any vindiction – we are studying meaningless strings. Now we want to interpret the statements – this is semantics. A basic method of semantics in propositional logic is through modelling the truth of a statement.

Definition. A **truth assignment** on a set Λ is a map $f : \Lambda \to \{\top, \bot\}$.

Example. *A* **boolean function** *is a truth assignment, where $\Lambda = \{\top, \bot\}$. More generally, an n-ary boolean function is a map $f : \{\top, \bot\}^n \to \{\top, \bot\}$. It is common to define such a function by a truth table. For an n-ary boolean function, we form a table with $n+1$ columns, and 2^n rows. In each row, we fill* *in a particular truth assignment in $\{\top, \bot\}^n$, and in the last column, the value of image of the truth assignment under f. One may combine multiple n-ary truth functions into the same table for brevity. There are 2^{2^n} n-ary truth functions. Here we consider the truth tables of some important unary and binary propositions in the truth table below.*

x	y	$H_\wedge(x,y)$	$H_\vee(x,y)$	$H_\Rightarrow(x,y)$	$H_\Leftrightarrow(x,y)$	$H_\neg(x)$
$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$
$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\bot$

If we have a homomorphism between two rings R and S, we may extend the map to a homomorphism between the corresponding polynomial rings $R[X]$ and $S[X]$. As with polynomials, we may extend any truth assignment f on Λ to a function f_* on $\text{SL}(\Lambda)$. This is done recursively. Let f be an arbitrary truth assignment. We define, recursively,

$$f_*(\neg s) = H_\neg(f_*(s)) \qquad f_*(s \circ w) = H_\circ(f_*(s), f_*(w))$$

The fact that this is well defined, and is defined for all elements of $\text{SL}(\Lambda)$, is clear from our discussion of syntax.

Definition. A statement $s \in \text{SL}(\Lambda)$ is a **tautology**, if, for any truth assignment f on Λ, $f_*(s) = \top$. A **contadiction** satisfies $f_*(s) = \bot$ for all s, and a **cotingent statement** is a statement which is neither a contradiction nor a tautology. We summarize the statement "s is a tautology" by $\models s$. We say a statement s **logically implies** w, written $s \models w$, if $\models s \Rightarrow w$, or correspondingly, if $f_*(w) = \top$ whenever $f_*(s) = \top$.

Suppose that we wish to verify whether $s \in \text{SL}(\Lambda)$ is a tautology. We say a variable $A \in \Lambda$ **occurs** in s if it is an element of the string. There can only be a finite number of variables $x_1, \dots, x_n$ which occur in s. Define a boolean function $g : \{0,1\}^n \to \{0,1\}$. By defining

$$g(y_1 \dots, y_n) = f_*^{(y_1, \dots, y_n)}(s)$$

where $f^{(y_1,\dots,y_n)}$ is the truth assignment formed by mapping x_i to y_i. This is well defined, because if h and k are two truth assignments which agree on the A_i, then they agree at s. Then one verifies that s is a tautology if and only if $g(y_1,\dots,y_n) = \top$, for all choices of x_i. This is verified by a simple use of a truth table.

Example. *For instance, for any variable $A \in \Lambda$, $A \vee \neg A$ is a tautology, $A \wedge \neg A$ is a contradiction, and $\neg A$ is contingent, which is verified by the truth table*

A	$A \vee \neg A$	$A \wedge \neg A$	$\neg A$
$\top$	$\top$	$\bot$	$\bot$
$\bot$	$\top$	$\bot$	$\top$

The first formula represents what is called the **law of excluded middle***, the second* **reductio ad absurdum***.*

Theorem 2.5 (Semantic Modus Ponens)**.** *If $\models s$ and $s \models w$, then $\models w$.*

Proof. Let f be a truth assignment. Then $f_*(s) = \top$ and

$$f_*(s \Rightarrow w) = H_{\Rightarrow}(f_*(s), f_*(w)) = H_{\Rightarrow}(\top, f_*(w)) = \top$$

This holds only when $f_*(w) = \top$. □

Definition. Let $x = (x_1, x_2, \dots, x_n)$ be a finite sequence of variables, and $s = (s_1, \dots, s_n)$ a sequence of statements in $\mathrm{SL}(\Lambda)$. If $w \in \mathrm{SL}(\Lambda)$ We shall define the substitution on $\mathrm{SL}(\Lambda)$, denoted $w[s_1/x_1, \dots, s_n/x_n]$, or $w[s/x]$ recursively by the formulae

$$x_j[s/x] = s_j \qquad y[s/x] = y \qquad \text{If } y \in \Lambda, y \notin x$$

$$(u \circ w)[s/x] = (u[s/x] \circ w[s/x]) \qquad (\neg w)[s/x] = (\neg w[s/x])$$

Theorem 2.6. *If $\models w$, then $\models w[s/x]$*

Proof. Consider a truth assignment f. We shall define another truth assignment $\tilde{f}$ such that $f(v[s/x]) = \tilde{f}(v)$ for all v. Define $\tilde{f}(x_i) = f(s_i)$, and if $y \notin x$, define $\tilde{f}(y) = f(y)$. Our base case, where v is a variable, satisfies the claim by construction. Then, by induction, if $v = (u \circ w)$, then

$$\tilde{f}_*(v) = H_\circ(\tilde{f}_*(u), \tilde{f}_*(w)) = H_\circ(f_*(u[s/x]), f_*(w[s/x])) = f_*(v[s/x])$$

A similar proof answers the case where $u = (\neg v)$, whic proves the claim. Thus if w is a tautology, then

$$f_*(w[s/x]) = \tilde{f}_*(w) = \top$$

So $w[s/x]$ is a tautology. □

Corollary 2.7. *If $v \vDash w$, then $v[s/x] \vDash w[s/x]$.*

2.3 Truth Functional Completeness

We would hope that our propositional logic can model all notions of truth. We shall call this truth functional completeness.

Definition. Let Λ be a set of boolean functions. The **clone** of Λ is the smallest set containing Λ, containing all projections $\pi^n_k : \{0,1\}^n \to \{0,1\}$,

$$\pi^n_k(x_1, \dots, x_n) = x_k$$

And such that if $g : \{0,1\}^n \to \{0,1\}$ is in the clone, and $f_1, \dots, f_n : \{0,1\}^m \to \{0,1\}$ are in the clone, then so is $g(f_1, \dots, f_n) : \{0,1\}^m \to \{0,1\}$. Λ is **truth functionally complete** if its clone is the set of all boolean functions.

Example. *$\{H_\neg, H_\wedge, H_\vee\}$ is a truth functionally complete, since every formula can be put in* **conjunctive normal form**, *for we may write the 'true and false' functions*

$$\top(x) = \top = H_\vee(H_\neg(x), x) \qquad \bot(x) = \bot = H_\wedge(H_\neg(x), x)$$

and then we write

$$f(x_1, \dots, x_n) = \bigvee_{\substack{(y_1,\dots,y_n) \in \{0,1\}^n \\ f(y_1,\dots,y_n) = \top}} \bigwedge_{i=1}^n H_\Leftrightarrow(x_i, y_i)$$

where we identify an element of $\{\top, \bot\}$ with the constant boolean function, $\bigcirc_{i=1}^{n} f_i = H_\circ(f_n, \bigcirc_{i=1}^{n-1})$, and

$$H_\Leftrightarrow(x,y) = H_\wedge(H_\Rightarrow(x,y), H_\Rightarrow(y,x)) = H_\wedge(H_\vee(H_\neg(x),y), H_\vee(H_\neg(y),x))$$

This can be further reduced, by noticing that

$$H_\wedge(x,y) = H_\neg(H_\vee(H_\neg(x), H_\neg(y)))$$

which is a truth functional form of Boole's inequality, implying $\{\neg, \vee\}$ is truth functionally complete. We can also consider **disjunctive normal form**.

Example. *The mathematician Henry M. Sheffer showed that there is a singleton set of truth functional connectives which is truth functionally complete. Consider the* **Sheffer stroke** *$x|y$, also known as* **NAND**, *and defined by the truth table*

x	y	$H_\vert(x,y)$
$\bot$	$\bot$	$\top$
$\bot$	$\top$	$\top$
$\top$	$\bot$	$\top$
$\top$	$\top$	$\bot$

Then $H_\neg(x) = H_|(x,x)$, and $H_\vee(x,y) = H_|(H_\neg(x), H_\neg(y))$, which implies (since this set is truth functionally complete), that the sheffer stroke is truth functionally complete.

The previous example is incredibly important to circuit design, which represents boolean functions with circuits. Since all truth functions can be built from the sheffer stroke, we need only make a circuit for the sheffer stroke, and we may build all other circuits by combining sheffer strokes. This is why computers consist of NAND gates.

Chapter 3

And Now, Computability

What is the fundamental notion of computation? Until the early 20th century little had been done to address this fundamental hole in logic. It is often the case that precise definitions of intuitive notions give rise to easy proofs of the most surprising consequence. In 1931, Kurt Gödel found the surprising fact that all sufficiently complicated axiomatic systems could talk of theorems which could neither be proved nor disproved. But a fundamental question remained – how was one to find proofs of theorems which could be proved or disproved. This was a problem with a long history of consideration – Leibniz had visualized his mathematical system as a calculational tool (hence the name), for which one should be able to design a tool which can put in equations and return answers. Babbage wanted such a tool to alleviate human errors in calculations, and devoted a large portion of his life to design a calculational machine, which he called the analytical engine. Nonetheless, it took a theoretical analysis to truly understand the nature of computation, and to understand its limitations. In the 30s, the logicians Alonzo Church and Alan Turing defined precise models of computation, and showed that there was no computational procedure for finding the proof of an algorithm. Fifty years later, 'theoretical computers' had become a common reality, and have changed fundamentally modern life. In this course, we shall describe mathematical models of computation which have been developed to analyze the limitations of various computational methods.

So how do we define precisely what a 'computational procedure' is? This was the major breakthrough of Church and Turing. Philisophically, one should be able to define this notion without reference to a computer, for humans have been computing far longer than microchips have. Nonetheless, Our model should probably take as model some physical object, for one needs some physical mechanism to compute anything (either electronically, or mentally). For applications and practicality, one should make this machine analogous to a industrial machine, since we understand these much more precisely than the brain. It is hoped that these models accurately reflect what a brain can do, and in the past 80 years no-one has found a procedure that a brain can do that our models cannot, so for now, it is a safe assumption.

We will begin by analyzing the automaton, a model of computation without stored memory. We will expand the amount of expression of the automaton by considering context-free grammars. Finally, we add memory by considering a Turing machine. It is the Church Turing theses that this is the ultimate model of computation – any real world computation can be modelled as an action on a Turing machine. From this model, and with the hypothesis of Church and Turing, we can make precise, philisophically interesting statements about the nature of computation.

As in mathematical logic, the objects of study are strings of symbols over a certain alphabet. One may understand the notion of computation syntactically, and the structure of strings give light to semantic facts about computation. The main idea of computability theory is that a mental computation can be modelled as a **decision problem** – find a computational model which will 'accept' certain strings over an alphabet (we will define this more precisely later). Most, if not all problems can be modelled in this form. Suppose our problem is to add two numbers. We are given a and b. The decision problem related to this problem is, if given a, b, and c, determine if $a + b = c$ is a correct result. Our symbol set is $\{0, 1, \dots, 9, :\}$, and we wish to model a computation which accepts all strings of the form "$a : b : c$", where a, b, and c are decimal strings for which $a + b = c$.

A subset of the set of all strings over an alphabet will be known as a **language**, and the goal of a decision problem is to design a machine which will accept only strings of the form above. Thus the goal of understanding computation reduces to studying the structure of languages. As a more dynamic discipline, we need more operations on strings to obtain languages from other languages. We need concatenation, but also **reversal**, the reversal of a string s will be denoted s^R. These operations are extended to languages by applying the operations on a component by component basis:

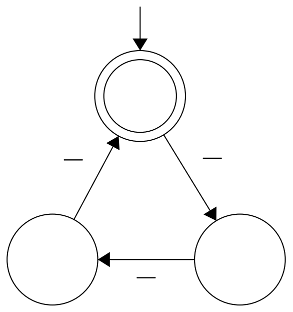
$$S \circ W = \{s \circ w : s \in S, w \in W\} \qquad S^R = \{s^R : s \in S\}$$

A **palindrome** is a string s for which $s^R = s$.

Chapter 4

Finite State Automata

Our initial model of computability is a computer with a very limited amount of memory. Surprisingly, we will still be able to compute a great many things with it. The idea of this model rests on a state diagram. Suppose we would like to describe an algorithm describing if a number is divisible by three. We shall represent a number by a string of dashes. For instance, "$-----$" represents the number 5. We describe the algorithm in a flow chart below



The algorithm proceeds as follows. Begin at the top node. For each dash, proceed clockwise aruond the triangle. If, at the end of our algorithm, we end up back at the top node, then the number is divisible by three. The basic idea of the finite automata is to describe computation via these flow charts – we follow a string around a diagram, and if we end up at a 'accept state', then we accept the string.

Definition. A **deterministic finite state automaton** is a 5-tuple $(Q, \Sigma, \Delta, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\Delta : Q \times \Sigma \to Q$ is the transition function, q_0 is the start state, and $F \subset Q$ are the accept states.

How does a finite state automaton 'work'. Draw a directed graph whose nodes are Q, and draw an edge between $s \in Q$ and $\Delta(s, \lambda)$, for each $\lambda \in \Sigma$. Take a string $s \in \Sigma^*$. We begin at the start state q_0. Sequentially, for each symbol in s, we take our current state, and using the transition function Δ move to the next state based on the symbol we are on at s. If, after we have gone through every symbol, we are at a state in F, then s is an 'accepted' string. Formally, what we are doing is extending Δ to $Q \times \Sigma^*$. We define $\Delta(\varepsilon, q) = q$, and, for $t \in \Sigma$, $\Delta(ts, q) = \Delta(t, \Delta(s, q))$.

Definition. Let $M = (Q, \Sigma, \Delta, q_0, F)$ be a finite state automata. Then M **accepts** a string s in Σ^* if $\Delta(s, q_0) \in F$. We call the set of all accepting strings the **language** of M, and denote the set as $L(M)$. A subset of Σ^* which is a language of a deterministic finite state automata is known as a **regular language**.

Example. *Consider $\Sigma = \{-\}$, as we have previously considered. Then the set of all 'dashes divisible by three' is regular, as we have shown. Formally, if we take $Q = \mathbf{Z}_3$, $\Delta(x, -) = x + 1$, $q_0 = 0$, and $F = \{0\}$, then $(Q, \Sigma, \Delta, q_0, F)$ recognizes the language. The 'graph' of the automata is exactly the graph we considered above.*

The set of regular languages is closed under conveniently many operations. We shall find enough operations that we can describe all regular languages as composed from basic languages from certain operations.

Theorem 4.1. *If $A, B \subset \Sigma^*$ are regular languages, then $A \cup B$ is regular.*

Proof. let $M = (Q, \Sigma, \Delta, q_0, F)$ and $N = (R, \Sigma, \Gamma, r_0, G)$ be automata recognizing A and B respectively. We shall define a finite automata recognizing

$A \cup B$. Let $S = Q \times R$, and $(\Delta \times \Gamma) : S \times \Sigma \to S$ by letting

$$(\Delta \times \Gamma)(q, r, \lambda) = (\Delta(q), \Gamma(r))$$

Let $H = \{(q,r) \in S : q \in F \text{ or } r \in G\}$. We contend $(S, \Sigma, \Delta \times \Gamma, (q_0, r_0), H)$ recognizes $A \cup B$, for, by induction, one verifies that $(\Delta \times \Gamma)(s, (q, r)) = (\Delta(s, q), \Gamma(s, r))$, and thus $(\Delta \times \Gamma)(s, (q_0, r_0)) \in H$ if and only if s is in $L(M)$ or s is in $L(N)$. □

We shall be less detailed in future proofs.

Theorem 4.2. *If A is a regular language, then A^c is regular.*

Proof. Take a machine accepting A, and take the complement of the accepting states. □

Corollary 4.3. *If A and B are regular languages, then $A \cap B$ are regular.*

It shall be convenient to expand our definition of automaton, in a manner which we shall repeat with more complicated models. It is a version of a finite automata that can parallel process. It will be notationally useful to define $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.

Definition. A **non-deterministic finite state automata** is $M = (Q, \Sigma, \Delta, q_0, F)$, where everything is the same except $\Delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ maps to possible states. We extend $\Delta : \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ by defining

$$\Delta(S, \varepsilon) = S \qquad \Delta(S, tv) = \Delta(t, \Delta(S, v))$$

Where we define, for an arbitrary function f, $f(S) = \{f(s) : s \in S\}$. The accepting language of M is $L(M) = \{s : \Delta(s, q_0)\}$