Metamathematics

Jacob Denson

February 4, 2016

Table Of Contents

Part I

Mathematical Logic

Chapter 1

What's Logic All About?

Metamathematics is the formal analysis of mathematical deduction and proof. Its aim is to understand the procedure and limitations of proof. Though studied since Plato's time, 'hard' metamathematics arose in response to paradoxes in early twentieth century mathematics. Each individual step of the paradoxes is intuitive, yet the conclusion ludicrous.

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

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

Russell's paradox rests on the following question: Is X in itself? If X was in X, then by definition, we would find that X cannot be contained in itself. So we are lead to believe that $X \notin X$. But then, by construction of X, we conclude X is in X after all! More colloquially, consider a town with a single barber, shaving 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. When we take X to be the set of all things, then $\mathcal{P}(X) \subset X$. There is an obvious injection of $\mathcal{P}(X)$ in X.*

Example (Richard)**.** *There are countably many english expressions, yet uncountable amounts of real numbers. Some real numbers cannot be described in english. Consider a particular enumeration containing all describable real numbers. Then, take the expression*

> *"The number constructed from Cantor's diagonal argument on the enumeration considered."*

Such a number cannot be described in english, yet is describable in english.

Example (Löb)**.** *Consider the Preposition B, defined to be true when* $B \implies A$ *is true. 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)**.** *Consider*

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

Then $(C \in C)$ *holds if and only if* $(C \in C) \implies P(C)$ *holds. We must have* $C \in C$*, for if* $C \notin C$*, then* $(C \in C) \implies P(C)$ *holds vacantly. But this implies* $(C \in C) \implies P(C)$*, so* $P(C)$ *is true. We conclude* $P(C)$ *is true, irrespective of the content of the statement* $P(C)$*.*

Resolving these paradoxes requires an acute understanding of proof, which only metamathematics can be provide. Which logical procedures summarize the construction of paradoxes? What standards do we use to define a mathematical statement? Our only hope to fix these issues is to apply the precise weapon of mathematical rigor. In the face of adversity, we do what mathematicians do best – define and conquer.

Defining fundamental principles of a field is rarely possible, and mathematical logic is no exception. Intuition is the only being who can completely grasp these principles; our concious minds will never obtain final definitions. Nonetheless, some properties are commonly accepted; it is these we use to understand the principle. A physicists's models are ideals, carved from reality in all senses but experimental parameters. No model describes a systems evolution exactly, and it is myopic to suggest a model is perfect. In spite of this, physics still does a bloody good job! In metamathematics, we discuss a 'model' of mathematical principles. Some principles are pinned down, others lost. We hope this model has enough vitality to provide key insights into the proofs found in real life. Whether the method is successful it up to interpretation.

A source of confusion in physics is the stylistic treatment of assumptions as absolute facts. A physicist describes "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. Such use is unavoidable,

since the former description is viceral, the latter dry to the bone. Keep this principle in mind as we begin to study logic.

The whole idea of metamathematics may seem circular. How do we understand mathematics through mathematic's eyes? This is the reason for formality. By placing mathematical understanding in a model, formal mathematics becomes a second order object. Proofs about this system rely on intuition, whereas proofs in the system have static, precise formulations. In the conversion mathematics becomes stale, abstract, and unintuitive. Such is the price to pay for precise theorems about mathematics itself.

Chapter 2

Prepositional Logic

We shall begin with propositional logic, the simplest model of truth. To understand propositional logic, we construct a mathematical model, known as a **formal language**, which represents the language in which mathematics is performed. The formal language is then analyzed by common mathematics. The standard formal language for logic is an analysis of strings, sequences of abstract symbols from a given **alphabet**. Strings represent mathematical statements; manipulating these strings models how a mathematician infers some mathematical statement from another. It is best to see the tool in action to understand its utility, so we proceed swiftly.

2.1 Syntax

Each abstract symbol in an alphabet represents a precise form in colloquial speech. We begin with prepositional logic, which analyzes basic notions of truth and falsity. Some statements are **atomic**, because they cannot be divided into more base statements. "Socrates is a man" is an atomic statement, as is "every woman is human". "Socrates is a man and every woman is a human" is not atomic, for the statement consists of two separate statements, composed by the connective "and". In English, "every woman is a human" can be broken into statements like "Julie is a human", "Laura is a human", yet propositional logic still considers this statement as atomic; the model does not have the capability to model these complex statements, which are the realm of predicate logic, which we shall analyze in the next chapter.

Definition. Let Λ be a set. The set Λ^* of **strings over** Λ is all finite, possibly empty, sequences with elements in Λ.

The empty string is denoted ε. A string $(v_1, \ldots, v_n)$ is often denoted $v_1 v_2 \ldots v_n$. We shall identify an element in Λ with the corresponding one letter string in Λ^*. The concatenation of two strings $s = s_1 \ldots s_n$ and $w = w_1 \ldots w_m$, denoted sw, is the string $s_1 \ldots s_n w_1 \ldots w_m$. A **substring** of a string s is a string u such that $s = wuv$, for some strings w and v. Substrings are consequtive subsequences of characters in s. 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 disjoint from $\{(,), \wedge, \vee, \neg, \Rightarrow, \Leftrightarrow\}$. The **propositional language with atoms in** Λ, denoted $\mathrm{SL}(\Lambda)$, is the smallest subset of $(\Lambda \cup \{(,), \wedge, \vee, \neg, \Rightarrow, \Leftrightarrow\})^*$ such that

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

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

Each **connective** represents a certain linguistical form.

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"

Later on, the connection of symbols to meaning will become clear. For now, they are abstract symbols without intrinsic meaning.

Take care to notice that $\mathrm{SL}(\Lambda)$ is the *smallest* set constructed, in the same way that most 'smallest objects' exist in mathematics, because the

intersection of sets satisfying the set of statements defining the set also satisfy the statements. This property leads to the most useful proof method in logic.

Theorem 2.1 (Structural Induction)**.** *Consider a proposition that can be applied to $SL(\Lambda)$. Suppose the proposition 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. Let P be some property to consider. Note the set

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

satisfies axioms (1) and (2) which define $\mathrm{SL}(\Lambda)$, so $\mathrm{SL}(\Lambda) \subset K$. □

Consider the following application of the proof method.

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. □

We need a more in depth theorem, to correctly parse statements. What is the value of $2+7-5-4$? Is it, by collecting factors in fairs, $9-1=8$, or $9-9=0$. This is the reason for parenthesis. One splits mathematical logic between into syntax, studying strings, and semantics, interpreting strings. We are studying syntax to begin with, so that we may begin semantics. Equations must be understood before we calculate with them.

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. $wu = (\neg s)$: March through all cases. Suppose w has the same number of left brackets than right. w cannot equal $($ or $(\neg$, nor $(\neg v$, where v is a prefix of s; 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. $wu = (s \circ w)$: Continue the string march. w cannot equal $($, nor $(v$ by induction. Similarily, w cannot equal $(s\circ$, nor $(s \circ v$, where v is a substring of w, so w must equal $(s \circ w)$.

Careful analysis of each case also shows that w must have at least as many left brackets as right brackets. □

Corollary 2.4. *Every string in $SL(\Lambda)$ can be written uniquely as an atom Λ, or $(\neg s)$ and $(s \circ w)$, where s and w are elements of $SL(\Lambda)$. The unique connective is known as* **principal**.

Proof. Such representations trivially exist. 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 equal. If we have two representations $(\neg s) = (\neg w)$, then by chopping off symbols, $s = w$. It is impossible to have two distinct representations $(s \circ w) = (\neg u)$, for no element of $\mathrm{SL}(\Lambda)$ begins with $\neg$. 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 representations are unique. □

The corollary above allows us to construct recursive definitions. Let $\Lambda = \{A, B, C\}$, and $\Gamma = \{X, Y, Z\}$. We would like to consider $\mathrm{SL}(\Lambda)$ and $\mathrm{SL}(\Gamma)$ the same, by considering a natural bijection. We extend the map

$$X \mapsto A \quad Y \mapsto B \quad Z \mapsto C$$

by the definition

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

Such a map is well defined and injective, by the corollary.

Before we finish with syntax, it is interesting to discuss a less natural, but syntactically simpler method of forming sentences, called **polish**

notation, after its inventor Jan Łukasiewicz. Rather than writing connectives in **infix notation**, like $(u \wedge v)$ and $(u \Rightarrow v)$, we use **prefix notation**, like $\wedge uv$ and $\Rightarrow uv$. We do not need brackets to parse statements anymore, 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 understanding of what symbols mean. Now we want to interpret. A basic semantic method is to define whether a statement is 'true'.

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

Example. *An n-ary* **boolean function** *is a truth assignment, where* $\Lambda = \{\top, \bot\}^n$. *It is common to define such functions by truth tables. For n-ary boolean functions, 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.*

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

This table defines the boolean functions $H_\circ$.

Homomorphisms between two rings R and S, extend to homomorphisms between the polynomial rings $R[X]$ and $S[X]$. Similarily, we may extend truth assignments f on Λ to assignments f_* on $\mathrm{SL}(\Lambda)$. This is done recursively. Let f be an arbitrary truth assignment. Define

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

then f_* is defined on all of $\mathrm{SL}(\Lambda)$.

Definition. Let $s \in \mathrm{SL}(\Lambda)$ be a statement. s is a

1. **tautology**, if, for any truth assignment f on Λ, $f_*(s) = \top$.
2. **contadiction** if $f_*(s) = \bot$.
3. **cotingent statement** if s is neither a contradiction nor a tautology.
4. **satisfiable** if it is not a contradiction.

We summarize the statement "s is a tautology" by $\models s$. We say a statement s **semantically 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 $s \in \mathrm{SL}(\Lambda)$ is a tautology. We say a variable $x \in \Lambda$ **occurs** in s if it is an element of the string. There can only be a finite number of variables $x_1, \ldots, x_n$ which occur in s. Define a boolean function $g : \{0,1\}^n \to \{0,1\}$,

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

where $f^{(y_1,\ldots,y_n)}$ 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 x_i, then they agree at s. Then one verifies that s is a tautology if and only if $g(y_1, \ldots, y_n) = \top$ for all choices of the y_i. One then simply builds a truth table. To prevent errors, it is best to construct a truth table for the subformulas in the principal connective of a formula before constructing the truth table of the formula. This may be done side by side, in the same 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 is the **law of excluded middle**, *the second* **reductio ad absurdum**.

Example. *Let $A \vDash B$ be a tautology, and suppose that A and B have no variables in common. Then A is a contradiction, or B is a tautology, for if there is a truth assignment on the variables of A which make the statement true, and a truth assignment on B which cause the statement to be false, then we may combine the truth assignments to create a truth assignment in which $A \Rightarrow B$ is false.*

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

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 $x = (x_1, x_2, \ldots, x_n)$ be a finite sequence of variables, and $s = (s_1, \ldots, s_n)$ a sequence of statements. If $w \in \mathrm{SL}(\Lambda)$, we shall define **substitution**, denoted $w[s_1/x_1, \ldots, s_n/x_n]$, 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 $\vDash w$, then $\vDash w[s/x]$*

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 $u = (\neg v)$. Since w is a tautology,

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

So $w[s/x]$ is a tautology. □

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

Theorem 2.8 (Craig Interpolation)**.** *Let $A \models B$, and let the set of variables shared by A and B be $x_1, \ldots, x_n$. Then there is a statement C, known as the* **iterpolant***, containing only the variables x_i, such that $A \models C$ and $C \models B$.*

Proof. We proceed by induction on the number of variables in A which do not occur in B. If every variable in A occurs in B, let $C = A$. In the general case, fix some variable x in A but not in B, take y in both A and B, and define

$$D = A[(y \wedge \neg y)/x] \vee A[(y \vee \neg y)/x]$$

If $f_*(A) = \top$, and $f_*(x) = \bot$, then $f_*(A[(y \wedge \neg y)/x]) = \top$. If $f_*(x) = \top$, then $f_*(A[(y \vee \neg y)/x]) = \top$, so $A \models D$. In addition, $D \models B$. Let $f_*(D) = \top$. Then $f_*(A[(y \wedge \neg y)/x]) = \top$, or $f_*(A[(y \vee \neg y)/x]) = \top$. In the first case, we modify the truth assignment f so that $f_*(x) = \bot$ (without changing the values of B or D). Then $f_*(A) = \top$, so $f_*(B) = \top$. The other case follows by letting $f_*(x) = \top$. By induction, there is an iterpolant C for which $D \models C$ and $C \models B$, and then $A \models C$, since $(x \Rightarrow y) \wedge (y \Rightarrow z) \models x \Rightarrow z$. □

Example. *Consider $A = (y_1 \Rightarrow x) \wedge (x \Rightarrow y_2)$ and $B = (y_1 \wedge z) \Rightarrow (y_2 \wedge z)$. Then $A \models B$. We shall find an interpolant.*

$$\begin{aligned} C = &[(y_1 \Rightarrow (y_1 \vee \neg y_1)) \wedge ((y_1 \vee \neg y_1) \Rightarrow y_2)] \\ &\vee [(y_1 \Rightarrow (y_1 \wedge \neg y_1)) \wedge ((y_1 \wedge \neg y_1) \Rightarrow y_2)] \end{aligned}$$

This equation can be simplified to $\neg y_1 \vee y_2$.

2.3 Truth Functional Completeness

We hope that propositional logic can model all notions of truth. Here we argue why our logic can model all such notions.

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

$$\pi_k^n(x_1, \ldots, x_n) = x_k$$

and in addition, if $g : \{0,1\}^n \to \{0,1\}$ and $f_1, \ldots, f_n : \{0,1\}^m \to \{0,1\}$ are in the clone, then so is $g(f_1, \ldots, f_n) : \{0,1\}^m \to \{0,1\}$. 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

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

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

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

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**, *left to the reader to define.*

Example. *The mathematician Henry M. Sheffer found a single truth function which is truth functionally complete. Consider the* **Sheffer stroke** *$x|y$, also known as* **NAND**, *defined by the truth table*

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

Then $H_\neg(x) = H_\mid(x,x)$, and $H_\vee(x,y) = H_\mid(H_\neg(x), H_\neg(y))$, 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. Logical statements can be represented by boolean functions. Since all truth functions can be built from the sheffer stroke, we need only make an atomic circuit for the sheffer stroke, and then all other circuits are constructed by combining sheffer strokes. This is why computers consist of millions of NAND gates.

2.4 Deduction

When mathematicians want to derive whether a statement is true, they do not construct truth functions. Instead, they argue *why* the statement is true. Here we shall provide the mechanics for modelling a mathematical argument. We will show that truth tables and arguments are equivalent – a statement is a tautology if and only if it can be proved.

Definition. The set of thorems of $\mathrm{SL}(\Lambda)$ is the smallest set such that for any $s, w, u \in \mathrm{SL}(\Lambda)$,

1. Any axiom is a theorem, where the axioms are the statements

$$s \Rightarrow (w \Rightarrow s)$$

$$(s \Rightarrow (w \Rightarrow u)) \Rightarrow ((s \Rightarrow w) \Rightarrow (s \Rightarrow u))$$

$$(\neg s \Rightarrow \neg w) \Rightarrow ((\neg s \Rightarrow w) \Rightarrow s)$$

   The axioms are known as (A1), (A2), and (A3) respectively.

2. Modus Ponens holds in our system. If $s \Rightarrow w$ and s are theorems, then w is a theorem. When we apply modus ponens, we may write that the theorem was obtained by (MP).

We shall write $\vdash s$ to state that s is a theorem.

For statements, the 'smallness' characterization gives us structural induction. For theorems, we obtain the notion of a 'proof'. A statement s is a theorem of $\mathrm{SL}(\Lambda)$ if and only if there is a sequence of formulae $(s_1, \ldots, s_n)$ such that $s_n = s$, and each s_i is either an axiom, or is obtained from some s_j and s_k by modus ponens, where $j, k < i$. This sequence is known as a proof. Often, we list a proof from top to bottom, where we reference how we obtained each element of the sequence alongside the proof.

Example. *Let us construct a proof of* $\vdash s \Rightarrow s$*, for any* $s \in SL(\Lambda)$.

$\vdash \mathsf{s} \Rightarrow \mathsf{s}$	
1. $(\mathsf{s} \Rightarrow ((\mathsf{s} \Rightarrow \mathsf{s}) \Rightarrow \mathsf{s}))$	*(A1)*
2. $(\mathsf{s} \Rightarrow ((\mathsf{s} \Rightarrow \mathsf{s}) \Rightarrow \mathsf{s})) \Rightarrow ((\mathsf{s} \Rightarrow (\mathsf{s} \Rightarrow \mathsf{s})) \Rightarrow (\mathsf{s} \Rightarrow \mathsf{s}))$	*(A2)*
3. $((\mathsf{s} \Rightarrow (\mathsf{s} \Rightarrow \mathsf{s})) \Rightarrow (\mathsf{s} \Rightarrow \mathsf{s}))$	*(1),(2),(MP)*
4. $(\mathsf{s} \Rightarrow (\mathsf{s} \Rightarrow \mathsf{s}))$	*(A1)*
5. $\mathsf{s} \Rightarrow \mathsf{s}$	*(3),(4),(MP)*

In future proofs, we shall be able to use $\vdash s \Rightarrow s$ *implicitly, since we now know the statement is a theorem. We will denote its application by* (I).

In other proof systems, we add additional axioms, and prove results from these axioms. If Γ is a subset of $\mathrm{SL}(\Lambda)$, then we may consider each member of Γ to be an axiom. We write $\Gamma \vdash s$ if one can prove s assuming all elements of Γ are axioms. That is, we may write a sequence $(s_1, \ldots, s_n)$, where $s_n = s$, and each s_i is either an axiom, an element of Γ, or is obtained by modus ponens from previous elements of the sequence.

Theorem 2.9 (Deduction Theorem)**.** *If* $\Gamma \cup \{s\} \vdash w$*, then* $\Gamma \vdash s \Rightarrow w$.

Proof. We prove the theorem by induction of the size of the proof of w. Consider a particular proof $(s_1, \ldots, s_n)$ of w from $\Gamma \cup \{s\}$. Suppose that $n = 1$. Then $s_1 = w$, and w must either be an axiom, an element of Γ, or equal to s. In the first and second case, the proof is equally valid in Γ, and so $\Gamma \vdash s \Rightarrow w$ follows from the axiom $(w \Rightarrow (s \Rightarrow w))$. If $s = w$, Then we have shown that $\Rightarrow w \Rightarrow w$, so obviously $\Gamma \vdash s \Rightarrow w$.

Now we consider the problem proved for $m < n$. $s_n = w$ is either an axiom, an element of Γ, equal to s, or proved by modus ponens from $s_i = (u \Rightarrow w)$ and $s_j = u$. We have already shown all but the last case. By induction, $\Gamma \vdash s \Rightarrow (u \Rightarrow w)$ and $\Gamma \vdash s \Rightarrow u$. But

$$(s \Rightarrow (u \Rightarrow w)) \Rightarrow ((s \Rightarrow u) \Rightarrow (s \Rightarrow w))$$

is an axiom, so $\Gamma \vdash s \Rightarrow w$. □

Example. *For any statements* s *and* w, $\{s \Rightarrow w, w \Rightarrow u\} \vdash s \Rightarrow u$*. This follows from a basic application of (A2). But this implies the two cut rules, that*

$$\vdash (s \Rightarrow w) \Rightarrow ((w \Rightarrow u) \Rightarrow (s \Rightarrow u))$$

$$\vdash (w \Rightarrow u) \Rightarrow ((s \Rightarrow w) \Rightarrow (s \Rightarrow u))$$

which is much more tricky to prove.

Example. *Let us prove double negation elimination* $(\neg\neg E)$, $\vdash \neg\neg s \Rightarrow s$, *by proving* $\neg\neg s \vdash s$.

$\neg\neg s \Rightarrow s$		
	1. $\neg\neg s$	
	2. $((\neg s \Rightarrow \neg\neg s)) \Rightarrow ((\neg s \Rightarrow \neg s) \Rightarrow s)$	*(A3)*
	3. $(\neg\neg s) \Rightarrow (\neg s \Rightarrow \neg\neg s)$	*(A1)*
	4. $(\neg s \Rightarrow \neg\neg s)$	*(1),(3),(MP)*
	5. $(\neg s \Rightarrow \neg s) \Rightarrow s$	*(2),(4),(MP)*
	6. $\neg s \Rightarrow \neg s$	*(I)*
	7. s	*(5),(6),(MP)*
8. $\neg\neg s \Rightarrow s$		*(1-7),(DT)*

Now lets prove $\vdash s \Rightarrow \neg\neg s$.

$s \Rightarrow \neg\neg s$		
1. $\neg\neg\neg s \Rightarrow \neg s$		$(\neg\neg E)$
	2. s	
	3. $(\neg\neg\neg s \Rightarrow \neg s) \Rightarrow ((\neg\neg\neg s \Rightarrow s) \Rightarrow \neg\neg s)$	*(A3)*
	4. $(\neg\neg\neg s \Rightarrow s) \Rightarrow \neg\neg s$	*(1), (3), (MP)*
	5. $s \Rightarrow (\neg\neg\neg s \Rightarrow s)$	*(A1)*
	6. $\neg\neg\neg s \Rightarrow s$	*(2),(5),(MP)*
	7. $\neg\neg s$	*(4),(6),(MP)*
8. $s \Rightarrow \neg\neg s$		*(2-7), (DT)*

Example. *Lets prove* $\neg w \vdash w \Rightarrow u$*, by proving* $\neg w, w \vdash u$*.*

	$\neg \mathsf{w} \Rightarrow (\mathsf{w} \Rightarrow \mathsf{u})$	
	1. $\neg \mathsf{w}$	
	2. w	
	3. $(\neg \mathsf{u} \Rightarrow \neg \mathsf{w}) \Rightarrow ((\neg \mathsf{u} \Rightarrow \mathsf{w}) \Rightarrow \mathsf{u})$	*(A3)*
	4. $\neg \mathsf{w} \Rightarrow (\neg \mathsf{u} \Rightarrow \neg \mathsf{w})$	*(A1)*
	5. $\neg \mathsf{u} \Rightarrow \neg \mathsf{w}$	*(1),(4), (MP)*
	6. $(\neg \mathsf{u} \Rightarrow \mathsf{w}) \Rightarrow \mathsf{u}$	*(3),(5),(MP)*
	7. $\mathsf{w} \Rightarrow (\neg \mathsf{u} \Rightarrow \mathsf{w})$	*(A1)*
	8. $\neg \mathsf{u} \Rightarrow \mathsf{w}$	*(2),(7),(MP)*
	9. u	*(6),(8),(MP)*
	10. $\mathsf{w} \Rightarrow \mathsf{u}$	*(2-9),(DT)*
	11. $\neg \mathsf{w} \Rightarrow (\mathsf{w} \Rightarrow \mathsf{u})$	*(1-10), (DT)*

This is a proof of **the law of contradiction**, *denoed* (LC).

Example. *Consider the following proof.*

	$(\mathsf{s} \Rightarrow \mathsf{w}) \Rightarrow (\neg \mathsf{w} \Rightarrow \neg \mathsf{s})$	
	1. $\mathsf{s} \Rightarrow \mathsf{w}$	
	2. $\neg \mathsf{w}$	
	3. $\neg \mathsf{w} \Rightarrow (\neg \neg \mathsf{s} \Rightarrow \neg \mathsf{w})$	*(A1)*
	4. $\neg \neg \mathsf{s} \Rightarrow \neg \mathsf{w}$	*(2),(3),(MP)*
	5. $(\neg \neg \mathsf{s} \Rightarrow \neg \mathsf{w}) \Rightarrow ((\neg \neg \mathsf{s} \Rightarrow \mathsf{w}) \Rightarrow \neg \mathsf{s})$	*(A3)*
	6. $(\neg \neg \mathsf{s} \Rightarrow \mathsf{w}) \Rightarrow \neg \mathsf{s}$	*(4),(5),(MP)*
	7. $\neg \neg \mathsf{s}$	
	8. $\neg \neg \mathsf{s} \Rightarrow \mathsf{s}$	$(\neg \neg E)$
	9. s	*(7),(8),(MP)*
	10. w	*(1),(9),(MP)*
	11. $\neg \neg \mathsf{s} \Rightarrow \mathsf{w}$	*(7-10), (DT)*
	12. $\neg \mathsf{s}$	*(6),(11), (MP)*
	13. $\neg \mathsf{w} \Rightarrow \neg \mathsf{s}$	*(2),(12),(MP)*
	11. $(\mathsf{s} \Rightarrow \mathsf{w}) \Rightarrow (\neg \mathsf{w} \Rightarrow \neg \mathsf{s})$	*(1-10), (DT)*

This is the **law of contraposition** (LCP).

Example. *Lets prove* $\vdash (s \Rightarrow w) \Rightarrow ((\neg s \Rightarrow w) \Rightarrow w)$.

$(\mathsf{s} \Rightarrow \mathsf{w}) \Rightarrow ((\neg \mathsf{s} \Rightarrow \mathsf{w}) \Rightarrow \mathsf{w})$	
1. $\mathsf{s} \Rightarrow \mathsf{w}$	
2. $(\mathsf{s} \Rightarrow \mathsf{w}) \Rightarrow (\neg \mathsf{w} \Rightarrow \neg \mathsf{s})$	*(LCP)*
3. $\neg \mathsf{w} \Rightarrow \neg \mathsf{s}$	*(1),(2),(MP)*
4. $\neg \mathsf{s} \Rightarrow \mathsf{w}$	
5. $(\neg \mathsf{w} \Rightarrow \neg \neg \mathsf{s})$	*(4),(LCP)*
6. $\neg \neg \mathsf{s} \Rightarrow \mathsf{s}$	$(\neg \neg E)$
7. $(\neg \mathsf{w} \Rightarrow \neg \neg \mathsf{s}) \Rightarrow ((\neg \neg \mathsf{s} \Rightarrow \mathsf{s}) \Rightarrow (\neg \mathsf{w} \Rightarrow \mathsf{s}))$	*(CUT)*
8. $(\neg \neg \mathsf{s} \Rightarrow \mathsf{s}) \Rightarrow (\neg \mathsf{w} \Rightarrow \mathsf{s})$	*(5),(7),(MP)*
9. $\neg \mathsf{w} \Rightarrow \mathsf{s}$	*(6),(8),(MP)*
10. $(\neg \mathsf{w} \Rightarrow \neg \mathsf{s}) \Rightarrow ((\neg \mathsf{w} \Rightarrow \mathsf{s}) \Rightarrow \mathsf{w})$	*(A3)*
11. $(\neg \mathsf{w} \Rightarrow \mathsf{s}) \Rightarrow \mathsf{w}$	$(3), (10), (MP)$
13. $(\neg \mathsf{s} \Rightarrow \mathsf{w}) \Rightarrow \mathsf{w}$	$(2-9), (DT)$
14. $(\mathsf{s} \Rightarrow \mathsf{w}) \Rightarrow ((\neg \mathsf{s} \Rightarrow \mathsf{w}) \Rightarrow \mathsf{w})$	$(1-10), (DT)$

This essentially proves that $(s \vee \neg s) \Rightarrow w$ *implies* w.

We shall verify that proofs never lead to contradiction.

Theorem 2.10. *If* $\vdash s$*, then* $\models s$.

Proof. This is a trivial proof by structural induction. Prove that all axioms are tautologies, and that the set of tautologies are closed under modus ponens. □

This theorem shows that there are some statements which are not provable in our system. In fact, if s is provable, then $\neg s$ is not provable. We call an axiom like this **absolutely consistant**. We shall show that all tautologies are provable, which shows the system is **complete**.

Lemma 2.11. *Let* $x_1, \ldots, x_n \in \Lambda$ *be variables in* $s \in SL(\Lambda)$*. Let* f *be a truth assignment, and define* $s' = s$ *if* $f_*(s) = \top$*, or* $s' = \neg s$ *if* $f_*(s) = \bot$*. Then*

$$x_1', \ldots, x_n' \vdash s'$$

Proof. We prove by structural induction. If $s = x_1$, then $x_1' = s'$, and $s \vdash s$ is a trivial theorem. If $s = \neg w$, we consider two cases. If $w' = w$, then

$s' = \neg\neg w$, and we have already shown $w \vdash \neg\neg w$, hence $x'_1, \ldots, x'_n \vdash s'$. If $w' = \neg w$, then $s' = w'$, and the theorem is trivial. If $s = w \Rightarrow u$, then either $f_*(w) = \top$ and $f_*(u) = \top$, or $f_*(w) = \bot$. In the first case, we have $x'_1, \ldots, x'_n \vdash u$, from which $x'_1, \ldots, x'_n \vdash w \Rightarrow u$ follows. In the second case, $x'_1, \ldots, x'_n \vdash \neg w$, and we have $\neg w \vdash (w \Rightarrow u)$. □

Corollary 2.12 (Completeness Theorem)**.** *If* $\vDash s$, $\vdash s$.

Proof. Let $x_1, \ldots, x_n$ be the variables in s. By the last lemma, we have

$$x_1, \ldots, x_n \vdash s \qquad x_1, \ldots, \neg\neg x_n \vdash s$$

By the deduction theorem

$$x_1, \ldots, x_{n-1} \vdash x_n \Rightarrow s \qquad x_1, \ldots, x_{n-1} \vdash \neg x_n \Rightarrow s$$

But then, since $\vdash (x_n \Rightarrow s) \Rightarrow ((\neg x_n \Rightarrow s) \Rightarrow s)$,

$$x_1, \ldots, x_{n-1} \vdash s$$

By induction, we find $\vdash s$. □

Before we finish our discussion of semantics, we note that there are many other axioms systems which can be used to define a propositional calculus (in the sense that they prove all tautologies). Of most interest to us is the axiom system whose only connective is the sheffer stroke, and whose only axiom schema is

$$(B|(C|D))|[\{E|(E|E)\}|[(F|C)|((B|F)|(B|F))]]$$

and whose rule of inference is to infer D from $B|(C|D)$, and B. Of course, it is outlandish to attempt proofs in such a system.

2.5 Multivalued Logics and Independence

In classical logic, there are two truth values, $\bot$ and $\top$. We build proof systems which 'preserves' truth. That is, all axioms are tautologies, and anything we prove from these axioms are tautologies. However, there is nothing stopping us from consider a system with n truth values $\{T_0, T_1, \ldots, T_n\}$.

This may represent the degree of truth of a statement, or something subtler.Intuitionistic logic discards the law of contradiction. It distinguishes between statements which are justifiable – that is, they are provable, and those statements which are never false, and these can be considered 'multiple values' of truth. When creating proof systems for multi-valued logics, we hope the value of a statement which follows from other statements is less than or equal to the value of the original statement. For instance, consider Sorites paradox

(1) "A grain of sand is not a heap of sand".
(2) "Adding a grain of sand to something which is not a heap does not make it into a heap"
(3) "A grain cannot be turned into a heap by adding grains."

We fix this argument by considering it in a multi-valued logic system. For each k, consider the statements

($1.k$) "k grains of sand do not make a heap".
($2.k$) "Adding a grain to k grains of sand doesn't make $k+1$ grains a heap"
($3.k$) "$k+1$ grains do not make a heap."

Provided we choose the truth of $(2.k)$ to be less than the truth of $(1.k)$, when the truth of $(3.k)$ is less than the truth of $(1.k)$, then this argument is invalid. We are only allowed to conclude $(3.k)$ from $(1.k)$ and $(2.k)$ when $(3.k)$ is 'less true' than $(1.k)$ and $(2.k)$. As $k \to \infty$, this implies that the statement eventually becomes completely false, unless we have infintely many truth values. This book is not intended to talk about non-standard logics in detail. Instead, we use the notion of multivalued logic to attack the notion of independance. Are the axioms of our Hilbert system minimal – do some of the axioms apply some of the other axioms?

Let us start by discarding (A1) from our axiom system. We may only derive results from (A2), (A3), and (MP). Can we derive (A1) using these rules? We shall try and set a value system on this proof system which satisfies the decreasing property of proofs. We shall consider 'truth assignments' based on the table

x	y	$H_{\Rightarrow}(x,y)$	$H_{\neg}(x)$
0	0	0	1
0	1	2	1
0	2	2	1
1	0	2	1
1	1	2	1
1	2	0	1
2	0	0	0
2	1	0	0
2	2	0	0

With this definition, we may consider truth values of formulae. Note that $x \Rightarrow (y \Rightarrow x)$ evaluates to 2 when x is 1 and y is 2. We shall show any provable formula from (A2) and (A3) always takes the value zero. One calculates that this is true of any instance of (A2) and (A3). Furthermore, if s and $s \Rightarrow w$ is always zero, then w is always zero. But this shows that $x \Rightarrow (y \Rightarrow x)$ can never be proven using (A2) and (A3).

Chapter 3

First Order Logic

Remember our conclusion that “Julie is a human” and “Laura is a human” from the general statement “All women are human” in the first chapter? We shall attempt to formalize manipulation of these statements.

Part II

Computability

What is computation? Until the early 20th century, little had been done to address this fundamental hole in mathematical logic. It is often the case that precise definitions give rise to easy proofs of the most surprising consequence. In 1931, Kurt Gödel proved all sufficiently complicated axiomatic systems had unprovable theorems, but a fundamental question remained; how do we decide whether a theorem can be proved? It took a decade for Alonzo Church and Alan Turing to deduce the impossibility of such a claim. Fifty years later, 'theoretical computers' had become a common reality. We shall describe mathematical models of computation which have been developed to analyze the limitations of various computational methods.

Turing and Church's major breakthrough was precisely defining a 'computational procedure'. Philosophically, one should be able to define a procedure without reference to a computer, for humans computed long before microchips. On the other hand, models should reflect physical reality, since one needs a physical mechanism in order to computer, whether electronic or mental. It is hoped that these models accurately reflect the limitations of computational models, and in the past 80 years no-one has disproved this claim.

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 can also understand the notion of computation syntactically. 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. Suppose our problem is to add two numbers. We are given a and b. The analgous decision problem decides, given a, b, and c, whether $a + b = c$. Our symbol set is $\{0, 1, \ldots, 9, :\}$, 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 accepts only strings in the specified language. The goal of understanding computation reduces a languages structure.

As a more dynamic discipline than mathematical logic, we need more operations on strings to obtain languages from other languages. We obviously need concatenation, but also **reversal**, which will be denoted s^R. These operations are extended to languages by applying the operations on a component by component basis:

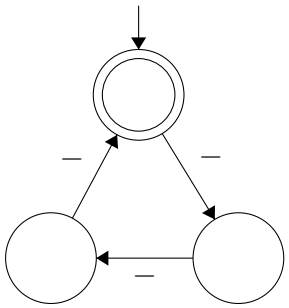
$$S \circ W = \{s \circ w : s \in S, w \in W\} \qquad S^R = \{s^R : s \in S\}$$

A **palindrome** is a string s for which $s^R = s$. If Σ is a set of strings, we shall let $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.

Chapter 4

Finite State Automata

Our initial model of computability is a computer with severely limited memory. Surprisingly, we shall still be able to compute a great many things. 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 around 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 **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, q_0 is the start state, $F \subset Q$ are the accept states, and

1. In a **deterministic finite state automata**, $\Delta : Q \times \Sigma \to Q$ is the transition function.
2. In a **non-deterministic finite state automata**, $\Delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the non-deterministic transition function.

We shall begin by discussing deterministic automata. A finite state machine 'works' exactly how our original algorithm worked. 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 linked by the symbol in s we are currently at. 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 deterministic 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 conveniently closed under many operations. We shall find enough such 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 $L(N)$. □

In future proofs, we shall be less detailed in constructing machines.

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

Proof. Take the complement of the accepting states in a machine. □

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

4.1 Non Deterministic Automata

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. This is the use of the non-deterministic automaton. We shall a say a state t is ε **reachable** from u if there is a sequence of states $t_0, t_1, \ldots, t_n$ with $t_0 = t$, $t_n = u$, and $t_{i+1} \in \Delta(t_i, \varepsilon)$. The set of all such states will temporarily be denoted $E(u)$.

Definition. In a non-deterministic finite state automata, we **accept** a string s if $s = q_1 \ldots q_n$, where each $q_i \in \Sigma_\varepsilon$, and a sequence of states $t_0, \ldots, t_{n+1}$, with $q_0 = t_0$ and $t_n \in F$, such that $t_{k+1} \in \Delta(t_k, s_k)$. The set of accepting strings of a machine M form the language $L(M)$.

We draw a graph with nodes Q, and with directed edges v to w if $w \in \Delta(v, \Sigma_\varepsilon)$. We begin at q_0. For a string s, we attempt to find a path from q_0 to an accept state, by following edges whose corresponding symbol is in s (or whose symbol is ε, in which we get for free). The string is accepted if such a path is possible.

It seems to be a much more complicated procedure to find if a string is accepted by a non-deterministic automata, but it turns out that every non-deterministic automata can be converted into a deterministic automata. The proof relies on the fact that we may exploit the operations of non-determinism, described using power sets of a set.

Theorem 4.4. *A language recognized by a non-deterministic finite state automata is regular.*

Proof. Let $M = (Q, \Sigma, \Delta, q_0, F)$ be a non-deterministic automata. Let

$$N = (\mathcal{P}(Q), \Sigma, \Gamma, \{q_0\}, \{S \in \mathcal{P}(Q) : S \cap F \neq \varnothing\}$$

where

$$\Gamma(S, t) = \{u \in Q : \exists s \in S : u \in \Delta(s, t)\}$$

Suppose s is accepted by N. Then $\Delta(\{q_0\}, s) \cap F \neq \varnothing$. Thus $s = q_1 \dots q_n$, and there are states $t_0 \dots t_n$, with $t_0 = q_0$ and $t_{k+1} \in \Delta(t_k, q_k)$, and $q_n \in F$. Thus s is accepted by M. Conversely, if s is accepted by M, then $\Delta(\{q_0\}, s) \cap F \neq \varnothing$, so s is accepted by N. □

We may now emply non-deterministic automata to understand deterministic automata.

Theorem 4.5. *If A and B are regular languages, then $A \circ B$ is regular.*

Proof. Let $M = (Q, \Sigma, \Delta, q_0, F)$ and $N = (R, \Sigma, \Gamma, r_0, G)$ be languages accepting A and B respectively. Without loss of generality, assume Q is disjoint from R. Consider the machine

$$O = (Q \cup R, \Sigma, \Pi, q_0, G)$$

Define

$$\Pi(s, t) = \begin{cases} \{\Delta(s, t)\} & s \in Q, \text{ and } t \neq \varepsilon \text{ or } s \notin F \\ \{\Delta(s, t), r_0\} & s \in F, t = \varepsilon \\ \{\Gamma(s, t)\} & s \in R \\ \varnothing & \text{otherwise} \end{cases}$$

We verify $sw \in L(O)$ for $s \in L(M)$, $w \in L(N)$. Now $r_0 \in \Pi(q_0, s)$, since $\Delta(t, w) \subset \Pi(t, w)$ for all $t \in Q$, and if $\Pi(t, w) \cap F = \varnothing$, $r_0 \in \Pi(t, w)$ by the ε closure property. But, similarily, $\Pi(r_0, w) \cap G \neq \varnothing$, from which it follows that $\Pi(t, sw) \cap G \neq \varnothing$.

If $v \in L(O)$, there is some substring k such that $\Pi(q_0, k) = r_0$ (for this is the only path from q_0 to G). If k is the shortest string for which this occurs, then $k \in L(O)$, for either $k = \varepsilon$ □

Theorem 4.6. *If A is a regular language, A^* is regular.*

Proof. s □

4.2 Regular Expressions

Automata are equivalent to a much more classical notion of computation – regular expressions, which are much more easy to manipulate in some circumstances.

Definition. A **regular expression** over an alphabet Σ is the smallest subset of $(\Sigma \cup \{\varnothing, \cup, *, (,)\})^*$ such that

1. $\varnothing$ is a regular expression as are $s \in \Sigma^*$.
2. If s and w are regular expressions, then so are s^*, $(s \cup w)$, and sw.

Every regular expression describes a regular language in the following manner.

Definition. A string $s \in \Sigma^*$ is recognized by an regular expression t if

1. $t \in \Sigma^*$, and $s = t$.
2. $t = (t' \cup t'')$, and s is recognized by t' or by t''.
3. $t = t't''$, and $s = s's''$, where s' is recognized by t', and s'' is recognized by t''.
4. $t = t'*$, and $s = \varepsilon$ or $s = s_1 \dots s_n$, where s_i is recognized by t'.

The regular language corresponding to a regular expression t is $L(t)$, the set of strings recognized by t. The set of all regular expressions on a language will be denoted $R(\Sigma)$

It is a simple consequence of our discourse that every language recognized by a regular expression *is actually* a regular language. In fact, we can show that every regular language is described by a regular expression. We shall describe an algorithm for converting a finite state automaton to a regular expression. We shall make a temporary generalization, by allowing non deterministic finite automata to have regular expressions in their transition functions.

Definition. A generalized non-deterministic finite state automate is a 5-tuple $M = (Q, \Sigma, \Delta, q_0, F)$, where $\Delta : Q \times Q \to R(\Sigma)$. A string $s \in \Sigma^*$ is accepted by M if we may write $s = s_1 \dots s_n$, and there are a sequence of states $q_0, \dots, q_{n+1}$ where $q_n \in F$, and $\Delta(q_i, q_{i+1})$ recognizes s_i.

We shall also assume a generalized automaton has one one end state, and that there are no states connecting to the beginning state of the automaton, which we can always model a finite automaton to possess without affecting the accepting language.

Theorem 4.7. *Every generalized automaton is equivalent to a regular expression.*

Proof. We shall prove this theorem for the more general class of generalized automaton. If the generalized automaton has two states, then there is only one transition from start state to begin state, and this transition describes a regular expression for the automaton. We will reduce every automaton to this form by induction. Suppose an automaton has n states. Fix a state $q \in Q$, which is neither the beginning or accepting state. Define a new automaton

$$N = (Q - \{q\}, \Sigma, \Delta', q_0, F)$$

where $\Delta'(a,b) = (\Delta(a,b) \cup \Delta(a,q)\Delta(q,q)^*\Delta(q,b))$ Then N is equivalent to M, and has one fewer state, and is thus equivalent to a regular expression. □

Corollary 4.8. *Every regular language is described by a regular expression.*

4.3 Limitations of Finite Automata

We've discovered a menagerie of different problems we can solve with finite automata, but it has already been foreshadowed that better machines await. Here we attack problems which cannot be solved by finite automata.

Theorem 4.9 (Pumping Lemma)**.** *Let L be a regular language. Then there is a number p, called the pumping length, such that if $s \in L$ satisfies $|s| \geqslant p$, then we may write $s = wuv$, where $|u| > 0$, $|wu| \leqslant p$, and $wu^i v \in L$ for all $i \geqslant 0$.*

Proof. Apply pidgeonhole principle. □

Example. *$L = \{0^k 1 0^k : k \geqslant 0\}$ is not regular. If it was regular, it would have a pumping length p. Then $0^p 1 0^p$ is in L, so we may write $0^p 1 0^p = wuv$, where $|wu| < p$. Then $u = 0^k$, for $k > 0$, and $wv = 0^{p-k} 1 0^p \in L$, which is clearly not the case, contradicting regularity.*

derivable

Chapter 5

Context Free Languages

Finite state machines are not a very general method of c of variablesomputation. We would like our notion of computability to decide on a much more complicated set of problems. Thus we must define new classes of computable language.

Definition. A **Context Free Grammar** is (V, Σ, R, S), where V is a set of variables, Σ is a character set, disjoint from V, R is a relation between V and $(V \cup \Sigma)^*$, and $S \in V$ is the start variable.

We say a string $w \in (V \cup \Sigma)^*$ is **directly derivable** from $s \in (V \cup \Sigma)^*$ if $w = tvt'$, $s = tt''t'$ where $v \in V$, $t, t', t'' \in (V \cup \Sigma)^*$, and $(v, t'') \in R$. The smallest transitive relation containing the 'directly derivable' relation is the derivability relation. To reiterate, a string s is **derivable** from w if there is a sequence $s_0, \ldots, s_n$, with $s_0 = s$, $s_n = w$, and s_{i+1} is directly derivable from s_i. Such a sequence is known as a **derivation**. The language of a grammar is the set of all strings in Σ^* derivable from the start state S. As in finite automata, the language of a grammar G will be denoted $L(G)$. A language is **context-free** if it is the language of a context free grammar.

Lemma 5.1. *Let $G = (V, \Sigma, R, S)$ and $H = (V, \Sigma, R', S)$ be two different context free languages. If every $(v, s) \in R'$, is derivable in G, then $L(H) \subset L(G)$.*

Proof. If we can show that every direct derivation in H is a derivation in G, then all derivations in H are derivations in G, since derivations form

the smallest transitive relation containing the direct derivations in H, and the derivations in G certainly satisfy this. If w is directly derived from s in H, then there is a rule (B, u), $s = rBt$, and $w = rut$. There is a sequence $s_0 \dots s_n$ deriving u from B in G. But then $(rs_0t) \dots (rs_nt)$ is a derivation of w from s, so w is derivable from s in G. □

A string is **ambiguous** if it can be derived in two different ways. A grammar is ambiguous if its language contains an ambiguous string. Ambiguity is unfortunate when parsing a language, since it means we may be able to interpret the language in two different ways.

Example. *First order logic can be defined by an unambiguous grammar. To develop the language, we took a bottom up approach, but we can also take a top up approach. Consider any particular first order language* $FO(\Lambda, \{\mathcal{P}^n\}, \{\mathcal{F}^n\})$, *where* Λ *are variables,* $\mathcal{P}^n$ *are n-ary predicates, and* $\mathcal{F}^n$ *are n-ary functions. Then the terms of the language form a grammar. First, we must include function formation rules. For each* $f \in \mathcal{F}^n$, *and* $x \in \Lambda$ *we have the rules*

$$T \mapsto f(\underbrace{T, T, \dots, T}_{n \text{ times}}) | x$$

Then the terms form an unambiguous grammar, as we showed. The formulas are then defined by the rules, for each $P \in \mathcal{P}^n$ *and* $x \in \Lambda$, *by*

$$F \mapsto P(T, T, \dots, T) | T = T | (F \wedge F) | (F \vee F) | (\neg F) | (F \Rightarrow F) | (\forall x : F) | (\exists x : F)$$

Then this grammar is also unambiguous.

It took a lot of work to show that first order logic is unambiguous. This make sense, because in general it is impossible to decide whether a language is unambiguous.

Definition. A grammar (V, Σ, R, S) is in **Chomsky Normal Form** if the only relations in R are of the form (A, BC), where $A, B, C \in V$, $B, C \neq S$, or (A, a), where $a \in \Sigma$.

Theorem 5.2. *Every context-free language can be recognized by a context-free grammar in Chomsky normal form.*

Proof. We shall reduce any context free grammar $G_0 = (V, \Sigma, R_0, S)$ to a context free grammar in normal form. We may assume that any rule does not link to S, by adding an additional start variable which links to S. First, we shall remove any ε rules. Order the set of all variables in $V - \{S\}$:

$$A_1, \ldots, A_n$$

Recursively define grammars $G_n = (V, \Sigma, R_n, S)$, where

$$R_n = (R_{n-1} - \{(A_n, \varepsilon)\}) \cup \{(B : sw) : (B, sA_n w) \in R, sw \neq \varepsilon \text{ or } \neg(\exists i \leqslant n : B = A_i)\}$$

Then $L(G_n) = L(G_{n-1})$. Certainly $L(G_n) \subset L(G_{n-1})$ by the lemma, since any rule in G_n is derivable in G_{n-1}. Let $(s_0, \ldots, s_n)$ be a derivation with $s_0 = S$. □