FOUNDATIONS OF MATHEMATICS
CHAPTER 1: FOUNDATIONS OF GEOMETRY

1. Introduction

Plane geometry is an area of mathematics that has been studied since ancient times. The roots of the word *geometry* are the Greek words *ge* meaning "earth" and *metria* meaning "measuring". This name reflects the computational approach to geometric problems that had been used before the time of Euclid, (ca. 300 B.C.), who introduced the axiomatic approach in his book, *Elements*. He was attempting to capture the reality of the space of our universe through this abstraction. Thus the theory of geometry was an attempt to capture the essence of a particular model.

Euclid did not limit himself to plane geometry in the *Elements*, but also included chapters on algebra, ratio, proportion and number theory. His book set a new standard for the way mathematics was done. It was so systematic and encompassing that many earlier mathematical works were discarded and thereby lost for historians of mathematics.

We start with a discussion of the foundations of plane geometry because it gives an accessible example of many of the questions of interest.

2. Axioms of plane geometry

Definition 2.1. *The theory of* **Plane Geometry***, PG, has two one-place predicates, Pt and Ln, to distinguish the two kinds of objects in plane geometry, and a binary incidence relation, In, to indicate that a point is on or incident with a line.*
By an abuse of notation, write $P \in Pt$ for $Pt(P)$ and $\ell \in Ln$ for $Ln(\ell)$.
There are five axioms in the theory PG:

(A_0) *(Everything is either a point or line, but not both; only points are on lines.)*
$(\forall x)((x \in Pt \vee x \in Ln)\ \&\ \neg(x \in Pt\ \&\ x \in Ln))\ \&\ (\forall x, y)(xIny \to (x \in Pt\ \&\ y \in Ln))$.

(A_1) *(Any two points belong to a line.)*
$(\forall P, Q \in Pt)(\exists \ell \in Ln)(PIn\ell\ \&\ QIn\ell)$.

(A_2) *(Every line has at least two points.)*
$(\forall \ell \in Ln)(\exists P, Q \in Pt)(PIn\ell\ \&\ QIn\ell\ \&\ P \neq Q)$.

(A_3) *(Two lines intersect in at most one point.)*
$(\forall \ell, g \in Ln)(\forall P, Q \in Pt)((\ell \neq g\ \&\ P, QIn\ell\ \&\ P, QIng) \to P = Q)$.

(A_4) *(There are four points no three on the same line.)* $(\exists P_0, P_1, P_2, P_3 \in Pt)(P_0 \neq P_1 \neq P_2 \neq P_3\ \&\ P_2 \neq P_0 \neq P_3 \neq P_1\ \&\ (\forall \ell \in Ln)($
$\neg(P_0In\ell\ \&\ P_1In\ell\ \&\ P_2In\ell)\ \&$
$\neg(P_0In\ell\ \&\ P_1In\ell\ \&\ P_3In\ell)\ \&$
$\neg(P_0In\ell\ \&\ P_2In\ell\ \&\ P_3In\ell)\ \&$
$\neg(P_1In\ell\ \&\ P_2In\ell\ \&\ P_3In\ell))$.

The axiom labeled 0 simply says that our objects have the types we intend, and is of a different character than the other axioms. In addition to these axioms, Euclid had one that asserted the existence of circles of arbitrary center and arbitrary radius, and one that asserted that all right angles are equal. He also had another axiom for points and lines, called the parallel postulate, which he attempted to show was a consequence of the other axioms.

Definition 2.2. *Two lines are parallel if there is no point incident with both of them.*

Definition 2.3. *For $n \geq 0$, the n-parallel postulate, P_n, is the following statement:*

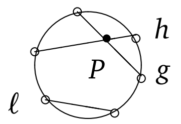
(P_n) *For any line ℓ and any point Q not on the line ℓ, there are n lines parallel to ℓ through the point Q.*

P_1 is the familiar parallel postulate.
For nearly two thousand years, people tried to prove what Euclid had conjectured. Namely, they tried to prove that P_1 was a consequence of the other axioms. In the 1800's, models of the other axioms were produced which were not models of P_1.

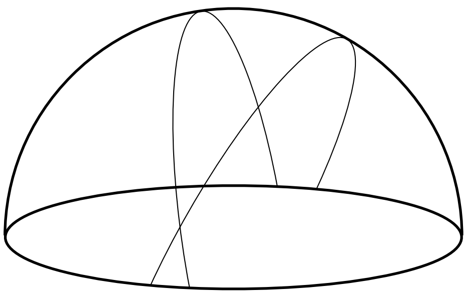
3. Non-Euclidean models

Nikolai Lobachevski (1793-1856), a Russian mathematician, and Janos Bolyai (1802-1860), a Hungarian mathematician, both produced models of the other axioms together with the parallel postulate P_∞, that there are infinitely many lines parallel to a given line through a given point. This geometry is known as *Lobachevskian Geometry*. It is enough to assume $P_{\geq 2}$ together with the circle and angle axioms to get P_∞.

Example 3.1. (A model for Lobachevskian Geometry): Fix a circle, C, in a Euclidean plane. The points of the geometry are the interior points of C. The lines of the geometry are the intersection of lines of the Euclidean plane with the interior of the circle. Given any line ℓ of the geometry and any point Q of the geometry which is not on ℓ, every Euclidean line through Q which intersects ℓ on or outside of C gives rise to a line of the geometry which is parallel to ℓ.



Example 3.2. (A model for Riemannian Geometry): Fix a sphere, S, in Euclidean 3-space. The points of the geometry may be thought of as either the points of the upper half of the sphere, or as equivalence classes consisting of the pairs of points on opposite ends of diameters of the sphere (antipodal points). If one chooses to look at the points as coming from the upper half of the sphere, one must take care to get exactly one from each of the equivalence classes. The lines of the geometry are the intersection of the great circles with the points. Since any two great circles meet in two antipodal points, every pair of lines intersects. Thus this model satisfies P_0.



Bernhard Riemann (1826-1866), a German mathematician, was a student of Karl Gauss (1777-1855), who is regarded as the greatest mathematician of the nineteenth century. Gauss made contributions in the areas of astronomy, geodesy and electricity as well as mathematics. While Gauss considered the possibility of non-Euclidean geometry, he never published anything about the subject.

4. Finite geometries

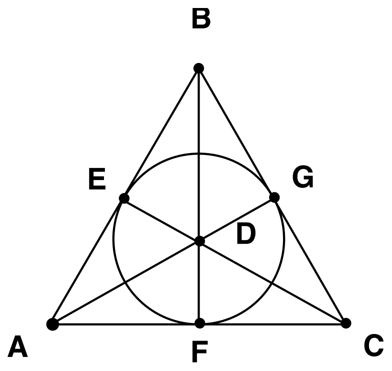
Next we turn to finite geometries, ones with only finitely many points and lines. To get the theory of the finite projective plane of order q, denoted PG(q), in addition to the five axioms given above, we add two more:

($A_5(q)$) Every line contains exactly $q+1$ points.
($A_6(q)$) Every point lies on exactly $q+1$ lines.

The first geometry we look at is the finite projective plane of order 2, PG(2), also known as the Fano Plane.

Theorem 4.1. *The theory PG(2) consisting of PG together with the two axioms $(5)_2$ and $(6)_2$ determines a finite geometry of seven points and seven lines, called the Fano plane.*

Proof. See Exercise **??** to prove from the axioms and Exercise **??** that the following diagram gives a model of PG(2) and that any model must have the designated number of points and lines. □



Next we construct a different model of this finite geometry using a vector space. The vector space underlying the construction is the vector space of dimension three over the field of two elements, $Z_2 = \{0, 1\}$. The points of the geometry are one dimensional subspaces. Since a one-dimensional subspace of Z_2 has exactly two triples in it, one of which is the triple $(0,0,0)$, we identify the points with the triples of 0's and 1's that are not all zero. The lines of the geometry are the two dimensional subspaces. The incidence relation is determined by a point is on a line if the one dimensional subspace is a subspace of the two dimensional subspace. Since a two dimensional subspace is picked out as the orthogonal complement of a one- dimensional subspace, each two dimensional subspace is identified with the non-zero triple, and to test if point (i,j,k) is on line $[\ell,m,n]$, one tests the condition

$$i\ell + jm + kn \equiv 0 \pmod 2.$$

There are exactly $2^3 = 8$ ordered triples of 0's and 1's, of which one is the all zero vector. Thus the ordered triples pick out the correct number of points and lines. The following array gives the incidence relation, and allows us to check that there are three points on every line and three lines through every point.

In	[1,0,0]	[0,1,0]	[0,0,1]	[1,1,0]	[1,0,1]	[0,1,1]	[1,1,1]
(1,0,0)	0	1	1	0	0	1	0
(0,1,0)	1	0	1	0	1	0	0
(0,0,1)	1	1	0	1	0	0	0
(1,1,0)	0	0	1	1	0	0	1
(1,0,1)	0	1	0	0	1	0	1
(0,1,1)	1	0	0	0	0	1	1
(1,1,1)	0	0	0	1	1	1	0

The vector space construction works over other finite fields as well. The next bigger example is the projective geometry of order 3, PG(3). The points are the one dimensional subspaces of the vector space of dimension 3 over the field of three elements, $Z_3 = \{0,1,2\}$. This vector space has $3^3 - 1 = 27 - 1 = 26$ non-zero vectors. Each one dimensional subspace has two non-zero elements, so there are $26/2 = 13$ points in the geometry. As above, the lines are the orthogonal or perpendicular complements of the subspaces that form the lines, so there are also 13 of them. The test for incidence is similar to the one above, except that one must work $\pmod 3$ rather than $\pmod 2$.

This construction works for each finite field. In each case the order of the projective geometry is the size of the field.

The next few lemmas list a few facts about projective planes.

Lemma 4.2. *In any model of PG(q), any two lines intersect in a point.*

Lemma 4.3. *In any model of PG(q) there are exactly $q^2 + q + 1$ points.*

Lemma 4.4. *In any model of PG(q) there are exactly $q^2 + q + 1$ lines.*

For models built using the vector space construction over a field of q elements, it is easy to compute the number of points and lines as $\frac{q^3-1}{q-1} = q^2 + q + 1$. However there are non-isomorphic projective planes of the same order. For a long time four non-isomorphic planes of order nine were known, each obtained by a variation on the above vector space construction. Recently it has been shown with the help of a computer that there are exactly four non-isomorphic planes of order nine.

Since the order of a finite field is always a prime power, the models discussed so far all have prime power order. Much work has gone into the search for models of non-prime power order. A well-publicized result showed that there were no projective planes of order 10. This proof required many hours of computer time.

FOUNDATIONS OF MATHEMATICS
CHAPTER 2: PROPOSITIONAL LOGIC

1. The basic definitions

Propositional logic concerns relationships between sentences built up from primitive proposition symbols with logical connectives.

The symbols of the language of predicate calculus are

(1) Logical connectives: $\neg$, $\&$, $\vee$, $\rightarrow$, $\leftrightarrow$
(2) Punctuation symbols: (,)
(3) Propositional variables: $A_0, A_1, A_2, \ldots$.

A propositional variable is intended to represent a proposition which can either be true or false. Restricted versions, $\mathscr{L}$, of the language of propositional logic can be constructed by specifying a subset of the propositional variables. In this case, let $\mathrm{PVar}(\mathscr{L})$ denote the propositional variables of $\mathscr{L}$.

Definition 1.1. *The collection of sentences, denoted $Sent(\mathscr{L})$, of a propositional language $\mathscr{L}$ is defined by recursion.*

(1) *The basis of the set of sentences is the set $\mathrm{PVar}(\mathscr{L})$ of propositional variables of $\mathscr{L}$.*
(2) *The set of sentences is closed under the following production rules:*
    (a) *If A is a sentence, then so is $(\neg A)$.*
    (b) *If A and B are sentences, then so is $(A \,\&\, B)$.*
    (c) *If A and B are sentences, then so is $(A \vee B)$.*
    (d) *If A and B are sentences, then so is $(A \rightarrow B)$.*
    (e) *If A and B are sentences, then so is $(A \leftrightarrow B)$.*

Notice that as long as $\mathscr{L}$ has at least one propositional variable, then $Sent(\mathscr{L})$ is infinite. When there is no ambiguity, we will drop parentheses.

In order to use propositional logic, we would like to give meaning to the propositional variables. Rather than assigning specific propositions to the propositional variables and then determining their truth or falsity, we consider truth interpretations.

Definition 1.2. *A* truth interpretation *for a propositional language $\mathscr{L}$ is a function*

$$I : \mathrm{PVar}(\mathscr{L}) \rightarrow \{0, 1\}.$$

If $I(A_i) = 0$, then the propositional variable A_i is considered represent a false proposition under this interpretation. On the other hand, if $I(A_i) = 1$, then the propositional variable A_i is considered to represent a true proposition under this interpretation.

There is a unique way to extend the truth interpretation to all sentences of $\mathscr{L}$ so that the interpretation of the logical connectives reflects how these connectives are normally understood by mathematicians.

Definition 1.3. *Define an extension of a truth interpretation* $I : \mathrm{PVar}(\mathscr{L}) \to \{0,1\}$ *for a propositional language to the collection of all sentences of the language by recursion:*

(1) *On the basis of the set of sentences,* $\mathrm{PVar}(\mathscr{L})$*, the truth interpretation has already been defined.*
(2) *The definition is extended to satisfy the following closure rules:*
   (a) *If* $I(A)$ *is defined, then* $I(\neg A) = 1 - I(A)$*.*
   (b) *If* $I(A)$ *and* $I(B)$ *are defined, then* $I(A \,\&\, B) = I(A) \cdot I(B)$*.*
   (c) *If* $I(A)$ *and* $I(B)$ *are defined, then* $I(A \vee B) = max\,\{I(A), I(B)\}$*.*
   (d) *If* $I(A)$ *and* $I(B)$ *are defined, then*

$$I(A \to B) = \begin{cases} 0 & \text{if } I(A) = 1 \text{ and } I(B) = 0, \\ 1 & \text{otherwise}. \end{cases} \tag{1}$$

   (e) *If* $I(A)$ *and* $I(B)$ *are defined, then* $I(A \leftrightarrow B) = 1$ *if and only if* $I(A) = I(B)$*.*

Intuitively, tautologies are statements which are always true, and contradictions are ones which are never true. These concepts can be defined precisely in terms of interpretations.

Definition 1.4. *A sentence* φ *is a* tautology *for a propositional language* $\mathscr{L}$ *if every truth interpretation* I *has value 1 on* φ*,* $I(\varphi) = 1$*.* φ *is a* contradiction *if every truth interpretation* I *has value 0 on* φ*,* $I(\varphi) = 0$*. Two sentences* φ *and* ψ *are* logically equivalent*, in symbols* $\varphi \Longleftrightarrow \psi$*, if every truth interpretation* I *takes the same value on both of them,* $I(\varphi) = I(\psi)$*. A sentence* φ *is* satisfiable *if there is some truth interpretation* I *with* $I(\varphi) = 1$*.*

The notion of logical equivalence is an equivalence relation; that is, it is a reflexive, symmetric and transitive relation. The equivalence classes given by logical equivalence are infinite for non-trivial languages (i.e., those languages containing at least one propositional variable). However, if the language has only finitely many propositional variables, then there are only finitely many equivalence classes.

Notice that if $\mathscr{L}$ has n propositional variables, then there are exactly $d = 2^n$ truth interpretations, which we may list as $\mathscr{I} = \{I_0, I_1, \ldots, I_{d-1}\}$. Since each I_i maps the truth values 0 or 1 to each of the n propositional variables, we can think of each truth interpretation as a function from the set $\{0, \ldots, n-1\}$ to the set $\{0, 1\}$. The collection of such functions can be written as $\{0,1\}^n$, which can also be interpreted as the collection of binary strings of length n.

Each sentence φ gives rise to a function $TF_\varphi : \mathscr{I} \to \{0,1\}$ defined by $TF_\varphi(I_i) = I_i(\varphi)$. Informally, TF_φ lists the column under φ in a truth table. Note that for any two sentences φ and ψ, if $TF_\varphi = TF_\psi$ then φ and ψ are logically equivalent. Thus there are exactly $2^d = 2^{2^n}$ many equivalence classes.

Lemma 1.5. *The following pairs of sentences are logically equivalent as indicated by the metalogical symbol* $\Longleftrightarrow$*:*

(1) $\neg\neg A \Longleftrightarrow A$.
(2) $\neg A \vee \neg B \Longleftrightarrow \neg(A \,\&\, B)$.
(3) $\neg A \,\&\, \neg B \Longleftrightarrow \neg(A \vee B)$.
(4) $A \to B \Longleftrightarrow \neg A \vee B$.
(5) $A \leftrightarrow B \Longleftrightarrow (A \to B) \,\&\, (B \to A)$.

Proof. Each of these statements can be proved using a truth table, so from one example the reader may do the others. Notice that truth tables give an algorithmic approach to questions of logical equivalence.

	A	B	$(\neg A)$	$(\neg B)$	$((\neg A)\vee(\neg B))$	$(A\,\&\,B)$	$(\neg(A\,\&\,B))$
I_0	0	0	1	1	1	0	1
I_1	1	0	0	1	1	0	1
I_2	0	1	1	0	1	0	1
I_3	1	1	0	0	0	1	0
					$\uparrow$		$\uparrow$

$\square$

Using the above equivalences, one could assume that $\neg$ and $\vee$ are primitive connectives, and define the others in terms of them. The following list gives three pairs of connectives each of which is sufficient to get all our basic list:

$$\neg, \vee$$

$$\neg, \ \&$$

$$\neg, \rightarrow$$

In logic, the word "theory" has a technical meaning, and refers to any set of statements, whether meaningful or not.

Definition 1.6. *A set Γ of sentences in a language $\mathscr{L}$ is satisfiable if there is some interpretation I with $I(\varphi)=1$ for all $\varphi\in\Gamma$. A set of sentences Γ logically implies a sentence φ, in symbols, $\Gamma\models\varphi$ if for every interpretation I, if $I(\psi)=1$ for all $\psi\in\Gamma$, then $I(\varphi)=1$. A (propositional) theory in a language $\mathscr{L}$ is a set of sentences $\Gamma\subseteq Sent(\mathscr{L})$ which is closed under logical implication.*

Notice that a theory as a set of sentences matches with the notion of the theory of plane geometry as a set of axioms. In studying that theory, we developed several models. The interpretations play the role here that models played in that discussion. Here is an example of the notion of logical implication defined above.

Lemma 1.7. $\{(A\,\&\,B),(\neg C)\}\models(A\vee B)$.

2. Disjunctive Normal Form Theorem

In this section we will show that the language of propositional calculus is sufficient to represent every possible truth function.

Definition 2.1.

(1) *A* literal *is either a propositional variable A_i or its negation $\neg A_i$.*
(2) *A* conjunctive clause *is a conjunction of literals and a* disjunctive clause *is a disjunction of literals. We will assume in each case that each propositional variable occurs at most once.*
(3) *A propositional sentence is in* disjunctive normal form *if it is a disjunction of conjunctive clauses and it is in* conjunctive normal form *if it is a conjunction of disjunctive clauses.*

Lemma 2.2.

(i) *For any conjunctive clause $C=\phi(A_1,\dots,A_n)$, there is a unique interpretation $I_C:\{A_1,\dots,A_n\}\to\{0,1\}$ such that $I_C(\phi)=1$.*
(ii) *Conversely, for any interpretation $I:\{A_1,\dots,A_n\}\to\{0,1\}$, there is a unique conjunctive clause C_I (up to permutation of literals) such that $I(C_I)=1$ and for any interpretation $J\neq I$, $J(C_I)=0$.*

Proof. (i) Let

$$B_i = \begin{cases} A_i & \text{if } C \text{ contains } A_i \text{ as a conjunct} \\ \neg A_i & \text{if } C \text{ contains } \neg A_i \text{ as a conjunct} \end{cases}.$$

It follows that $C = B_1 \ \& \ \ldots \ \& \ B_n$. Now let $I_C(A_i) = 1$ if and only if $A_i = B_i$. Then clearly $I(B_i) = 1$ for $i = 1, 2, \ldots, n$ and therefore $I_C(C) = 1$. To show uniqueness, if $J(C) = 1$ for some interpretation J, then $\phi(B_i) = 1$ for each i and hence $J = I_C$.

(ii) Let

$$B_i = \begin{cases} A_i & \text{if } I(A_i) = 1 \\ \neg A_i & \text{if } I(A_i) = 0 \end{cases}.$$

Let $C_I = B_1 \ \& \ \ldots \ \& \ B_n$. As above $I(C_I) = 1$ and $J(C_I) = 1$ implies that $J = I$.

It follows as above that I is the unique interpretation under which C_I is true. We claim that C_I is the unique conjunctive clause with this property. Suppose not. Then there is some conjunctive clause C' such that $I(C') = 1$ and $C' \neq C_I$. This implies that there is some literal A_i in C' and $\neg A_i$ in C_I (or vice versa). But $I(C') = 1$ implies that $I(A_i) = 1$ and $I(C_I) = 1$ implies that $I(\neg A_i) = 1$, which is clearly impossible. Thus C_I is unique. □

Here is the Disjunctive Normal Form Theorem.

Theorem 2.3. *For any truth function $F : \{0,1\}^n \to \{0,1\}$, there is a sentence ϕ in disjunctive normal form such that $F = TF_\phi$.*

Proof. Let $I_1, I_2, \ldots, I_k$ be the interpretations in $\{0,1\}^n$ such that $F(I_i) = 1$ for $i = 1, \ldots, k$. For each i, let $C_i = C_{I_i}$ be the conjunctive clauses guaranteed to hold by the previous lemma. Now let $\phi = C_1 \vee C_2 \vee \ldots \vee C_k$. Then for any interpretation I,

$$\begin{aligned} TF_\phi(I) = 1 & \text{ if and only if } I(\phi) = 1 \text{ (by definition)} \\ & \text{if and only if } I(C_i) = 1 \text{ for some } i = 1, \ldots, k \\ & \text{if and only if } I = I_i \text{ for some } i \text{ (by the previous lemma)} \\ & \text{if and only if } F(I) = 1 \text{ (by the choice of } I_1, \ldots, I_k) \end{aligned}$$

Hence $TF_\phi = F$ as desired. □

Example 2.4. *Suppose that we want a formula $\phi(A_1, A_2, A_3)$ such that $I(\phi) = 1$ only for the three interpretations $(0,1,0)$, $(1,1,0)$ and $(1,1,1)$. Then*

$$\phi = (\neg A_1 \ \& \ A_2 \ \& \ \neg A_3) \vee (A_1 \ \& \ A_2 \ \& \ \neg A_3) \vee (A_1 \ \& \ A_2 \ \& \ A_3).$$

It follows that the connectives $\neg, \&, \vee$ are sufficient to express all truth functions. By the deMorgan laws (2,3 of Lemma 2.5) $\neg, \vee$ are sufficient and $\neg, \wedge$ are also sufficient.

3. Proofs

One of the basic tasks that mathematicians do is proving theorems. This section develops the Propositional Calculus, which is a system rules of inference for propositional languages. With it one formalizes the notion of proof. Then one can ask questions about what can be proved, what cannot be proved, and how the notion of proof is related to the notion of interpretations.

The basic relation in the Propositional Calculus is the relation *proves* between a set, Γ of sentences and a sentence B. A more long-winded paraphrase of the relation "Γ proves B" is "there is a proof of B using what ever hypotheses are needed from Γ". This relation is denoted $X \vdash Y$, with the following abbreviations for special cases:

Formal Version:	$\Gamma \vdash \{B\}$	$\{A\} \vdash B$	$\emptyset \vdash B$
Abbreviation:	$\Gamma \vdash B$	$A \vdash B$	$\vdash B$

Let $\perp$ be a new symbol that we will add to our propositional language. The intended interpretation of $\perp$ is 'falsehood,' akin to asserting a contradiction.

Definition 3.1. *A* formal proof *or derivation of a propositional sentence ϕ from a collection of propositional sentences Γ is a finite sequence of propositional sentences terminating in ϕ where each sentence in the sequence is either in Γ or is obtained from sentences occurring earlier in the sequence by means of one of the following rules.*

(1) *(Given rule) Any $B \in \Gamma$ may be derived from Γ in one step.*
(2) *(&-Elimination) If $(A \,\&\, B)$ has been derived from Γ then either of A or B may be derived from Γ in one further step.*
(3) *($\vee$-Elimination) If $(A \vee B)$ has been derived from Γ, under the further assumption of A we can derive C from Γ, and under the further assumption of B we can derive C from Γ, then we can derive C from Γ in one further step.*
(4) *($\rightarrow$-Elimination) If $(A \rightarrow B)$ and A have been derived from Γ, then B can be derived from Γ in one further step.*
(5) *($\perp$-Elimination) If $\perp$ has been deduced from Γ, then we can derive any sentence A from Γ in one further step.*
(6) *($\neg$-Elimination) If $\neg\neg A$ has been deduced from Γ, then we can derive A from Γ in one further step.*
(7) *(&-Introduction) If A and B have been derived from Γ, then $(A \,\&\, B)$ may be derived from Γ in one further step.*
(8) *($\vee$-Introduction) If A has been derived from Γ, then either of $(A \vee B)$, $(B \vee A)$ may be derived from Γ in one further step.*
(9) *($\rightarrow$-Introduction) If under the assumption of A we can derive B from Γ, then we can derive $A \rightarrow B$ from Γ in one further step.*
(10) *($\perp$-Introduction) If $(A \,\&\, \neg A)$ has been deduced from Γ, then we can derive $\perp$ from Γ in one further step.*
(11) *($\neg$-Introduction) If $\perp$ has been deduced from Γ and A, then we can derive $\neg A$ from Γ in one further step.*

The relation $\Gamma \vdash A$ can now be defined to hold if there is a formal proof of A from Γ that uses the rules given above. The symbol $\vdash$ is sometimes called a *(single) turnstile*. Here is a more precise, formal definition.

Definition 3.2. *The relation $\Gamma \vdash B$ is the smallest subset of pairs (Γ, B) from $\mathscr{P}(Sent) \times Sent$ which contains every pair (Γ, B) such that $B \in \Gamma$ and is closed under the above rules of deduction.*

We now provide some examples of proofs.

Proposition 3.3. *For any sentences A, B, C*

(1) $\vdash A \rightarrow A$

(2) $A \to B \vdash \neg B \to \neg A$

(3) $\{A \to B, B \to C\} \vdash A \to C$

(4) $A \vdash A \vee B$ *and* $A \vdash B \vee A$

(5) $\{A \vee B, \neg A\} \vdash B$

(6) $A \vee A \vdash A$

(7) $A \vdash \neg\neg A$

(8) $A \vee B \vdash B \vee A$ *and* $A \,\&\, B \vdash B \,\&\, A$

(9) $(A \vee B) \vee C \vdash A \vee (B \vee C)$ *and* $A \vee (B \vee C) \vdash (A \vee B) \vee C$

(10) $(A \,\&\, B) \,\&\, C \vdash A \,\&\, (B \,\&\, C)$ *and* $A \,\&\, (B \,\&\, C) \vdash (A \,\&\, B) \,\&\, C$

(11) $A \,\&\, (B \vee C) \vdash (A \,\&\, B) \vee (A \,\&\, C)$ *and* $(A \,\&\, B) \vee (A \,\&\, C) \vdash A \,\&\, (B \vee C$

(12) $A \vee (B \,\&\, C) \vdash (A \vee B) \,\&\, (A \vee C)$ *and* $(A \vee B) \,\&\, (A \vee C) \vdash A \vee (B \,\&\, C)$

(13) $\neg(A \,\&\, B) \vdash \neg A \vee \neg B$ *and* $\neg A \vee \neg B \vdash \neg(A \,\&\, B)$

(14) $\neg(A \vee B) \vdash \neg A \,\&\, \neg B$ *and* $\neg A \,\&\, \neg B \vdash \neg(A \vee B)$

(15) $\neg A \vee B \vdash A \to B$ *and* $A \to B \vdash \neg A \vee B$

(16) $\vdash A \vee \neg A$

We give brief sketches of some of these proofs to illustrate the various methods.

Proof.

1. $\vdash A \to A$

1	$\mid\ A$	Assumption
2	$\mid\ A$	Given
3	$A \to A$	$\to$-Introduction (1-2)

3. $\{A \to B, B \to C\} \vdash A \to C$

1	$A \to B$	Given
2	$B \to C$	Given
3	$\mid\ A$	Assumption
4	$\mid\ B$	$\to$-Elimination 1,3
5	$\mid\ C$	$\to$-Elimination 2,4
6	$A \to C$	$\to$-Introduction 3-5

4. $A \vdash A \vee B$ and $A \vdash B \vee A$

1	A	Given
2	$A \vee B$	$\vee$-Introduction 1

1	A	Given
2	$B \vee A$	$\vee$-Introduction 1

5. $\{A \vee B, \neg A\} \vdash B$

1	$A \vee B$	Given
2	$\neg A$	Given
3	$\vert\ \underline{A}$	Assumption
4	$\vert\ A \,\&\, \neg A$	& -Introduction 2,3
5	$\vert\ \bot$	$\bot$-Introduction 4
6	$\vert\ B$	$\bot$-Elimination 5
7	$\vert\ \underline{B}$	Assumption
8	$\vert\ B$	Given
9	B	$\vee$-Elimination 1-8

6. $A \vee A \vdash A$

1	$A \vee A$	Given
2	$\vert\ \underline{A}$	Assumption
3	$\vert\ A$	Given
4	$\vert\ \underline{A}$	Assumption
5	$\vert\ A$	Given
6	A	$\vee$-Elimination 1-5

7. $A \vdash \neg\neg A$

1	A	Given
2	$\vert\ \underline{\neg A}$	Assumption
3	$\vert\ A \,\&\, \neg A$	&-Introduction 1,2
4	$\vert\ \bot$	$\bot$-Introduction 3
5	$\neg\neg A$	$\neg$-Introduction 1-4

8. $A \vee B \vdash B \vee A$ and $A \,\&\, B \vdash B \,\&\, A$

1	$A \vee B$	Given
2	$\mid \underline{A}$	Assumption
3	$\mid B \vee A$	$\vee$-Introduction 2
4	$\mid \underline{B}$	Assumption
5	$\mid B \vee A$	$\vee$-Introduction 2
6	$B \vee A$	$\vee$-Elimination 1-5

1	$A \,\&\, B$	Given
2	A	&-Elimination
3	B	&-Elimination
4	$B \,\&\, A$	&-Introduction 2-3

10. $(A \,\&\, B) \,\&\, C \vdash A \,\&\, (B \,\&\, C)$ and $A \,\&\, (B \,\&\, C) \vdash (A \,\&\, B) \,\&\, C$

1	$(A \,\&\, B) \,\&\, C$	Given
2	$A \,\&\, B$	&-Elimination 1
3	A	&-Elimination 2
4	B	&-Elimination 2
5	C	&-Elimination 1
6	$B \,\&\, C$	&-Introduction 4,5
7	$A \,\&\, (B \,\&\, C)$	&-Introduction 3,6

13. $\neg(A \,\&\, B) \vdash \neg A \vee \neg B$ and $\neg A \vee \neg B \vdash \neg(A \,\&\, B)$

1	$\neg(A \,\&\, B)$	Given
2	$A \vee \neg A$	Item 6
3	$\mid\ \underline{\neg A}$	Assumption
4	$\mid\ \neg A \vee \neg B$	$\vee$-Introduction 4
5	$\mid\ \underline{A}$	Assumption
6	$\mid\ B \vee \neg B$	Item 6
7	$\mid\ \mid\ \underline{\neg B}$	Assumption
8	$\mid\ \mid\ \neg A \vee \neg B$	$\vee$-Introduction 7
9	$\mid\ \mid\ \underline{B}$	Assumption
10	$\mid\ \mid\ A \,\&\, B$	& -Introduction 5,9
11	$\mid\ \mid\ \perp$	$\perp$-Introduction 1,10
12	$\mid\ \mid\ \neg A \vee \neg B$	Item 5
13	$\mid\ \neg A \vee \neg B$	$\vee$-Elimination 6-12
14	$\neg A \vee \neg B$	$\vee$-Elimination 2-13

1	$\neg A \vee \neg B$	Given
2	$\mid\ \underline{A \,\&\, B}$	Assumption
3	$\mid\ A$	& -Elimination 2
4	$\mid\ \neg\neg A$	Item 8, 3
5	$\mid\ \neg B$	Disjunctive Syllogism 1,4
6	$\mid\ B$	& -Elimination 2
7	$\mid\ \perp$	$\perp$-Introduction 5,6
8	$\neg(A \,\&\, B)$	Proof by Contradiction 2-7

15. $\neg A \vee B \vdash A \to B$ and $A \to B \vdash \neg A \vee B$

1	$\neg A \vee B$	Given
2	$\mid\ \underline{A}$	Assumption
3	$\mid\ \neg\neg A$	Item 8, 2
4	$\mid\ B$	Item 5 1,3
5	$A \to B$	$\to$-Introduction 2-4

1	$A \to B$	Given
2	$\vert\ \neg(\neg A \vee B)$	Assumption
3	$\vert\ \neg\neg A \,\&\, \neg B$	Item 14, 1
4	$\vert\ \neg\neg A$	&-Introduction 3
5	$\vert\ A$	$\neg$-Elimination 4
6	$\vert\ B$	$\to$-Elimination 1,5
7	$\vert\ \neg B$	&-Introduction 3
8	$\vert\ B \,\&\, \neg B$	&-Introduction 6,7
9	$\vert\ \bot$	$\bot$-Introduction 8
10	$\neg A \vee B$	$\bot$-Elimination 2-9

16. $\vdash A \vee \neg A$

1	$\vert\ \neg(A \vee \neg A)$	Assumption
2	$\vert\ \neg A \,\&\, \neg\neg A$	Item 14, 1
3	$\vert\ \bot$	$\bot$-Introduction 3
4	$\neg A \vee A$	$\neg\vee$-Rule (1-2)

□

The following general properties about $\vdash$ will be useful when we prove the soundness and completeness theorems.

Lemma 3.4. *For any sentences A and B, if $\Gamma \vdash A$ and $\Gamma \cup \{A\} \vdash B$, then $\Gamma \vdash B$.*

Proof. $\Gamma \cup \{A\} \vdash B$ implies $\Gamma \vdash A \to B$ by $\to$-Introduction. Combining this latter fact with the fact that $\Gamma \vdash A$ yields $\Gamma \vdash B$ by $\to$-Elimination. □

Lemma 3.5. *If $\Gamma \vdash B$ and $\Gamma \subseteq \Delta$, then $\Delta \vdash B$.*

Proof. This follows by induction on proof length. For the base case, if B follows from Γ on the basis of the Given Rule, then it must be the case that $B \in \Gamma$. Since $\Gamma \subset \Delta$ it follows that $B \in \Delta$ and hence $\Delta \vdash B$ by the Given Rule.

If the final step in the proof of B from Γ is made on the basis of any one of the rules, then we may assume by the induction hypothesis that the other formulas used in these deductions follow from Δ (since they follow from Γ). We will look at two cases and leave the rest to the reader.

Suppose that the last step comes by $\to$-Elimination, where we have derived $A \to B$ and A from Γ earlier in the proof. Then we have $\Gamma \vdash A \to B$ and $\Gamma \vdash B$. By the induction hypothesis, $\Delta \vdash A$ and $\Delta \vdash A \to B$. Hence $\Delta \vdash B$ by $\to$-Elimation.

Suppose that the last step comes from &-Elimination, where we have derived $A \,\&\, B$ from Γ earlier in the proof. Since $\Gamma \vdash A \,\&\, B$, by inductive hypothesis it follows that $\Delta \vdash A \,\&\, B$. Hence $\Delta \vdash B$ by &-elimination. □

Next we prove a version of the Compactness Theorem for our deduction system.

Theorem 3.6. *If* $\Gamma \vdash B$*, then there is a finite set* $\Gamma_0 \subseteq \Gamma$ *such that* $\Gamma_0 \vdash B$*.*

Proof. Again we argue by induction on proofs. For the base case, if B follows from Γ on the basis of the Given Rule, then $B \in \Gamma$ and we can let $\Gamma_0 = \{B\}$.

If the final step in the proof of B from Γ is made on the basis of any one of the rules, then we may assume by the induction hypothesis that the other formulas used in these deductions follow from some finite $\Gamma_0 \subseteq \Gamma$. We will look at two cases and leave the rest to the reader.

Suppose that the last step of the proof comes by $\vee$-Introduction, so that B is of the form $C \vee D$. Then, without loss of generality, we can assume that we derived C from Γ earlier in the proof. Thus $\Gamma \vdash C$. By the induction hypothesis, there is a finite $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \vdash C$. Hence by $\vee$-Introduction, $\Gamma_0 \vdash C \vee D$.

Suppose that the last step of the proof comes by $\vee$-Elimination. Then earlier in the proof

(i) we have derived some formula $C \vee D$ from Γ,
(ii) under the assumption of C we have derived B from Γ, and
(iii) under the assumption of D we have derived B from Γ.

Thus, $\Gamma \vdash C \vee D$, $\Gamma \cup \{C\} \vdash B$, and $\Gamma \cup \{D\} \vdash B$. Then by assumption, by the induction hypothesis, there exist finite sets Γ_0, Γ_1, and Γ_2 of Γ such that $\Gamma_0 \vdash C \vee D$, $\Gamma_1 \cup \{C\} \vdash B$ and $\Gamma_2 \cup \{D\} \vdash B$. By Lemma 3.5,

(i) $\Gamma_0 \cup \Gamma_1 \cup \Gamma_2 \vdash C \vee D$
(ii) $\Gamma_0 \cup \Gamma_1 \cup \Gamma_2 \cup \{C\} \vdash B$
(iii) $\Gamma_0 \cup \Gamma_1 \cup \Gamma_2 \cup \{D\} \vdash B$

Thus by $\vee$-Elimination, we have $\Gamma_0 \cup \Gamma_1 \cup \Gamma_2 \vdash B$. Since $\Gamma_0 \cup \Gamma_1 \cup \Gamma_2$ is finite and $\Gamma_0 \cup \Gamma_1 \cup \Gamma_2 \subseteq \Gamma$, the result follows. □

4. The Soundness Theorem

We now determine the precise relationship between $\vdash$ and $\models$ for propositional logic. Our first major theorem says that if one can prove something in A from a theory Γ, then Γ logically implies A.

Theorem 4.1 (Soundness Theorem). *If* $\Gamma \vdash A$*, then* $\Gamma \models A$*.*

Proof. The proof is by induction on the length of the deduction of A. We need to show that if there is a proof of A from Γ, then for any interpretation I such that $I(\gamma) = 1$ for all $\gamma \in \Gamma$, $I(A) = 1$.

(Base Case): For a one-step deduction, we must have used the Given Rule, so that $A \in \Gamma$. If the truth interpretation I has $I(\gamma) = 1$ for all $\gamma \in \Gamma$, then of course $I(A) = 1$ since $A \in \Gamma$.

(Induction): Assume the theorem holds for all shorter deductions. Now proceed by cases on the other rules. We prove a few examples and leave the rest for the reader.

Suppose that the last step of the deduction is given by $\vee$-Introduction, so that A has the form $B \vee C$. Without loss of generality, suppose we have derived B from Γ earlier in the proof. Suppose that $I(\gamma) = 1$ for all $\gamma \in \Gamma$. Since the proof of $\Gamma \vdash B$ is shorter than the given deduction of $B \vee C$, by the inductive hypothesis, $I(B) = 1$. But then $I(B \vee C) = 1$ since I is an interpretation.

Suppose that the last step of the deduction is given by &-Elimination. Suppose that $I(\gamma) = 1$ for all $\gamma \in \Gamma$. Without loss of generality A has been derived from a sentence of the form $A \,\&\, B$, which has been derived from Γ in a strictly shorter proof. Since $\Gamma \vdash A \,\&\, B$, it follows by inductive hypothesis that $\Gamma \models A \,\&\, B$, and hence $I(A \,\&\, B) = 1$. Since I is an interpretation, it follows that $I(A) = 1$.

Suppose that the last step of the deduction is given by $\rightarrow$-Introduction. Then A has the form $B \rightarrow C$. It follows that under the assumption of B, we have derived C from Γ. Thus $\Gamma \cup \{B\} \vdash C$ in a strictly shorter proof. Suppose that $I(\gamma) = 1$ for all $\gamma \in \Gamma$. We have two cases to consider.

Case 1: If $I(B) = 0$, it follows that $I(B \rightarrow C) = 1$.

Case 2: If $I(B) = 1$, then since $\Gamma \cup \{B\} \vdash C$,

it follows that $I(C) = 1$. Then $I(B \rightarrow C) = 1$. In either case, the conclusion follows. □

Now we know that anything we can prove is true. We next consider the contrapositive of the Soundness Theorem.

Definition 4.2. *A set* Γ *of sentences is* consistent *if there is some sentence* A *such that* $\Gamma \nvdash A$; *otherwise* Γ *is* inconsistent.

Lemma 4.3. Γ *of sentences is* inconsistent *if and only if there is some sentence* A *such that* $\Gamma \vdash A$ *and* $\Gamma \vdash \neg A$.

Proof. Suppose first that Γ is inconsistent. Then by definition, $\Gamma \vdash \phi$ for all formulas ϕ and hence $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ for *every* sentence A.

Next suppose that, for some A, $\Gamma \vdash A$ and $\Gamma \vdash \neg A$. It follows by &-Introduction that $\Gamma \vdash A \,\&\, \neg A$. By $\perp$-Introduction, $\Gamma \vdash \perp$. Then by $\perp$-Elimination, for each ϕ, $\Gamma \vdash \phi$. Hence Γ is inconsistent. □

Proposition 4.4. *If* Γ *is satisfiable, then it is consistent.*

Proof. Assume that Γ is satisfiable and let I be an interpretation such that $I(\gamma) = 1$ for all $\gamma \in \Gamma$. Now suppose by way of contradiction that Γ is *not* consistent. Then there is some sentence A such that $\Gamma \vdash A$ and $\Gamma \vdash \neg A$. By the Soundness Theorem, $\Gamma \models A$ and $\Gamma \models \neg A$. But then $I(A) = 1$ and $I(\neg A) = 1$ which is impossible since I is an interpretation. This contradiction demonstrates that Γ is consistent. □

In Section 5, we will prove the converse of the Soundness Theorem by showing that any consistent theory is satisfiable.

5. The Completeness Theorem

Theorem 5.1. *(The Completeness Theorem, Version I) If* $\Gamma \models A$, *then* $\Gamma \vdash A$.

Theorem 5.2. *(The Completeness Theorem, Version II) If* Γ *is consistent, then* Γ *is satisfiable.*

We will show that Version II implies Version I and then prove Version II. First we give alternate versions of the Compactness Theorem (Theorem 3.6).

Theorem 5.3. *(Compactness Theorem, Version II). If every finite subset of* Δ *is consistent, then* Δ *is consistent.*

Proof. We show the contrapositive. Suppose that Δ is not consistent. Then, for some B, $\Delta \vdash B \,\&\, \neg B$. It follows from Theorem 3.6 that Δ has a finite subset Δ_0 such that $\Delta_0 \vdash B \,\&\, \neg B$. But then Δ_0 is not consistent. □

Theorem 5.4. *(Compactness Theorem, Version III). Suppose that*

(i) $\Delta = \bigcup_n \Delta_n$,

(ii) $\Delta_n \subseteq \Delta_{n+1}$ *for every* n, *and*

(iii) Δ_n *is consistent for each* n.

Then Δ is consistent.

Proof. Again we show the contrapositive. Suppose that Δ is not consistent. Then by Theorem 5.4, Δ has a finite, inconsistent subset $F = \{\delta_1, \delta_2, \dots, \delta_k\}$. Since $\Delta = \bigcup_n \Delta_n$, there exists, for each $i \le k$, some n_i such that $\delta_i \in \Delta_i$. Letting $n = \max\{n_i : i \le k\}$, it follows from the fact that the Δ_j's are inconsistent that $F \subseteq \Delta_n$. But then Δ_n is inconsistent. □

Next we prove a useful lemma.

Lemma 5.5. *For any Γ and A, $\Gamma \vdash A$ if and only if $\Gamma \cup \{\neg A\}$ is inconsistent.*

Proof. Suppose first that $\Gamma \vdash A$. Then $\Gamma \cup \{\neg A\}$ proves both A and $\neg A$ and is therefore inconsistent.

Suppose next that $\Gamma \cup \{\neg A\}$ is inconsistent. It follows from $\neg$-Introduction that $\Gamma \vdash \neg\neg A$. Then by $\neg$-Elimination, $\Gamma \vdash A$. □

We are already in position to show that Version II of the Completeness Theorem implies Version I. We show the contrapositive of the statement of Version 1; that is, we show $\Gamma \nvdash A$ implies $\Gamma \not\models A$. Suppose it is not the case that $\Gamma \vdash A$. Then by Lemma 5.5, $\Gamma \cup \{\neg A\}$ is consistent. Thus by Version II, $\Gamma \cup \{\neg A\}$ is satisfiable. Then it is not the case that $\Gamma \models A$.

We establish a few more lemmas.

Lemma 5.6. *If Γ is consistent, then for any A, either $\Gamma \cup \{A\}$ is consistent or $\Gamma \cup \{\neg A\}$ is consistent.*

Proof. Suppose that $\Gamma \cup \{\neg A\}$ is inconsistent. Then by the previous lemma, $\Gamma \vdash A$. Then, for any B, $\Gamma \cup \{A\} \vdash B$ if and only if $\Gamma \vdash B$. Since Γ is consistent, it follows that $\Gamma \cup \{A\}$ is also consistent. □

Definition 5.7. *A set Δ of sentences is* maximally consistent *if it is consistent and for any sentence A, either $A \in \Delta$ or $\neg A \in \Delta$.*

Lemma 5.8. *Let Δ be maximally consistent.*

(1) *For any sentence A, $\neg A \in \Delta$ if and only if $A \notin \Delta$.*
(2) *For any sentence A, if $\Delta \vdash A$, then $A \in \Delta$.*

Proof. (1) If $\neg A \in \Delta$, then $A \notin \Delta$ since Δ is consistent. If $A \notin \Delta$, then $\neg A \in \Delta$ since Δ is maximally consistent.

(2) Suppose that $\Delta \vdash A$ and suppose by way of contradiction that $A \notin \Delta$. Then by part (1), $\neg A \in \Delta$. But this contradicts the consistency of Δ. □

Proposition 5.9. *Let Δ be maximally consistent and define the function $I : Sent \to \{0,1\}$ as follows. For each sentence B,*

$$I(B) = \begin{cases} 1 & \text{if } B \in \Delta; \\ 0 & \text{if } B \notin \Delta. \end{cases}$$

Then I is a truth interpretation and $I(B) = 1$ for all $B \in \Delta$.

Proof. We need to show that I preserves the four connectives: $\neg$, $\vee$, $\&$, and $\to$. We will show the first three and leave the last an exercise.

($\neg$): It follows from the definition of I and Lemma 5.8 that $I(\neg A) = 1$ if and only if $\neg A \in \Delta$ if and only if $A \notin \Delta$ if and only if $I(A) = 0$.

($\vee$): Suppose that $I(A \vee B) = 1$. Then $A \vee B \in \Delta$. We argue by cases. If $A \in \Delta$, then clearly $\max\{I(A), I(B)\} = 1$. Now suppose that $A \notin \Delta$. Then by completeness, $\neg A \in \Delta$. It follows from Proposition 3.3(5) that $\Delta \vdash B$. Hence $B \in \Delta$ by Lemma 5.8. Thus $\max\{I(A), I(B)\} = 1$.

Next suppose that $\max\{I(A), I(B)\} = 1$. Without loss of generality, $I(A) = 1$ and hence $A \in \Delta$. Then $\Delta \vdash A \vee B$ by $\vee$-Introduction, so that $A \vee B \in \Delta$ by Lemma 5.8 and hence $I(A \vee B) = 1$.

(&): Suppose that $I(A \,\&\, B) = 1$. Then $A \,\&\, B \in \Delta$. It follows from &-Elimination that $\Delta \vdash A$ and $\Delta \vdash B$. Thus by Lemma 5.8, $A \in \Delta$ and $B \in \Delta$. Thus $I(A) = I(B) = 1$.

Next suppose that $I(A) = I(B) = 1$. Then $A \in \Delta$ and $B \in \Delta$. It follows from &-Introduction that $\Delta \vdash A \,\&\, B$ and hence $A \,\&\, B \in \Delta$. Therefore $I(A \,\&\, B) = 1$. □

We now prove Version II of the Completeness Theorem.

Proof of Theorem 5.2. Let Γ be a consistent set of propositional sentences. Let $A_0, A_1, \ldots$ be an enumeration of the set of sentences. We will define a sequence $\Delta_0 \subseteq \Delta_1 \subseteq \ldots$ and let $\Delta = \bigcup_n \Delta_n$. We will show that Δ is a complete and consistent extension of Γ and then define an interpretation $I = I_\Delta$ to show that Γ is satisfiable.

$\Delta_0 = \Gamma$ and, for each n,

$$\Delta_{n+1} = \begin{cases} \Delta_n \cup \{A_n\}, & \text{if } \Delta_n \cup \{A_n\} \text{ is consistent} \\ \Delta_n \cup \{\neg A_n\}, & \text{otherwise.} \end{cases}$$

It follows from the construction that, for each sentence A_n, either $A_n \in \Delta_{n+1}$ or $\neg A_n \in \Delta_{n+1}$. Hence Δ is complete. It remains to show that Δ is consistent.

Claim 1: For each n, Δ_n is consistent.

Proof of Claim 1: The proof is by induction. For the base case, we are given that $\Delta_0 = \Gamma$ is consistent. For the induction step, suppose that Δ_n is consistent. Then by Lemma 5.6, either $\Delta_n \cup \{A_n\}$ is consistent, or $\Delta_n \cup \{\neg A_n\}$ is consistent. In the first case, suppose that $\Delta_n \cup \{A_n\}$ is consistent. Then $\Delta_{n+1} = \Delta_n \cup \{A_n\}$ and hence Δ_{n+1} is consistent. In the second case, suppose that $\Delta_n \cup \{A_n\}$ is inconsistent. Then $\Delta_{n+1} = \Delta_n \cup \{\neg A_n\}$ and hence Δ_{n+1} is consistent by Lemma 5.6.

Claim 2: Δ is consistent.

Proof of Claim 2: This follows immediately from the Compactness Theorem Version III.

It now follows from Proposition 5.9 that there is a truth interpretation I such that $I(\delta) = 1$ for all $\delta \in \Delta$. Since $\Gamma \subseteq \Delta$, this proves that Γ is satisfiable. □

We note the following consequence of the proof of the Completeness Theorem.

Theorem 5.10. *Any consistent theory* Γ *has a maximally consistent extension.* □

6. Completeness, Consistency and Independence

For a given set of sentences Γ, we sometimes identify Γ with the theory $Th(\Gamma) = \{B : \Gamma \vdash B\}$. Thus we can alternatively define Γ to be consistent if there is no sentence B such that $\Gamma \vdash B$ and $\Gamma \vdash \neg B$. Moreover, let us say that Γ is complete if for every sentence B, either $\Gamma \vdash B$ or $\Gamma \vdash \neg B$ (Note that if Γ is maximally consistent, it follows that Γ is complete, but the converse need not hold. We say that a consistent set Γ is *independent* if Γ has no proper subset Δ such that $Th(\Delta) = Th(\Gamma)$; this means that Γ is minimal among the sets Δ with $Th(\Delta) = Th(\Gamma)$.

For example, in the language $\mathscr{L}$ with three propositional variables A, B, C, the set $\{A, B, C\}$ is clearly independent and complete.

Lemma 6.1. Γ *is independent if and only if, for every* $B \in \Gamma$*, it is not the case that* $\Gamma - \{B\} \vdash B$.

Proof. Left to the reader. □

Lemma 6.2.
A set Γ of sentences is complete and consistent if and only if there is a unique interpretation I satisfied by Γ.

Proof. Left to the reader. $\square$

We conclude this chapter with several examples.

Example 6.3. *Let $\mathscr{L} = \{A_0, A_1, \dots\}$.*

(1) *The set $\Gamma_0 = \{A_0, A_0 \,\&\, A_1, A_1 \,\&\, A_2, \dots\}$ is complete but not independent.*
   — *It is complete since $\Gamma_0 \vdash A_n$ for all n, which determines the unique truth interpretation I where $I(A_n) = 1$ for all n.*
   — *It is not independent since, for each n, $(A_0 \,\&\, A_1 \cdots \,\&\, A_{n+1}) \to (A_0 \,\&\, \cdots \,\&\, A_n)$.*

(2) *The set $\Gamma_1 = \{A_0, A_0 \to A_1, A_1 \to A_2, \dots\}$ is complete and independent.*
   — *It is complete since $\Gamma_0 \vdash A_n$ for all n, which determines the unique truth interpretation I where $I(A_n) = 1$ for all n.*
   — *To show that Γ_1 is independent, it suffice to show that, for each single formula $A_n \to A_{n+1}$, it is not the case that $\Gamma_1 - \{A_n \to A_{n+1}\} \vdash (A_n \to A_{n+1})$. This is witnessed by the interpetation I where $I(A_j) = 1$ if $j \leq n$ and $I(A_j) = 0$ if $j > n$.*

(3) *The set $\Gamma_2 = \{A_0 \vee A_1, A_2 \vee A_3, A_4 \vee A_5, \dots\}$ is independent but not complete.*
   — *It is not complete since there are many different interpretations satisfied by Γ_2. In particular, one interpretation could make A_n true if and only if n is odd, and another could make A_n true if and only if n is even.*
   — *It is independent since, for each n, we can satisfy every sentence of Γ_2* except $A_{2n} \vee A_{2n+1}$ *by the interpretation I where $I(A_j) = 0$ exactly when $j = 2n$ or $j = 2n + 1$.*

FOUNDATIONS OF MATHEMATICS
CHAPTER 3: PREDICATE LOGIC

Propositional logic treats a basic part of the language of mathematics, building more complicated sentences from simple with connectives. However it is inadequate as it stands to express the richness of mathematics. Consider the axiom of the theory of **Plane Geometry, PG**, which expresses the fact that any two points belong to a line. We wrote that statement formally with two one-place predicates, *Pt* for points and *Ln* for lines, and one two-place predicate, *In* for incidence as follows:

$$(\forall P, Q \in Pt)(\exists \ell \in Ln)((P In \ell) \,\&\, (Q In \ell)).$$

This axiom includes predicates and quantifies certain elements. In order to test the truth of it, one needs to know how to interpret the predicates *Pt*, *Ln* and *In*, and the individual elements P, Q, ℓ. Notice that these elements are "quantified" by the quantifiers to "for every" and "there is ... such that." Predicate logic is an enrichment of propositional logic to include predicates, individuals and quantifiers, and is widely accepted as the standard language of mathematics.

1. The Language of Predicate Logic

The symbols of the language of the predicate logic are

(1) logical connectives, $\neg$, $\vee$, $\&$, $\rightarrow$, $\leftrightarrow$;
(2) the equality symbol $=$;
(3) predicate letters P_i for each natural number i;
(4) function symbols F_j for each natural number j;
(5) constant symbols c_k for each natural number k;
(6) individual variables v_ℓ for each natural number ℓ;
(7) quantifier symbols $\exists$ (the *existential* quantifier) and $\forall$ (the *universal* quantifer); and
(8) punctuation symbols (,).

A predicate letter is intended to represent a relation. Thus each predicate letter P is n-ary for some n, which means that we write $P(v_1, \ldots, v_n)$. Similarly, a function symbol also is n-ary for some n.

We make a few remarks on the quantifiers:

(a) $(\exists x)\phi$ is read "there exists an x such that ϕ holds."
(b) $(\forall x)\phi$ is read "for all x, ϕ holds."
(c) $(\forall x)\,\theta$ may be thought of as an abbreviation for $(\neg(\exists x)(\neg\theta))$.

Definition 1.1. *A countable first-order language is obtained by specifying a subset of the predicate letters, function symbols and constants.*

One can also work with uncountable first-order languages, but aside from a few examples in Chapter 4, we will primarily work with countable first-order languages. An example of a first-order language is the language of arithmetic.

Example 1.2. *The* language of arithmetic *is specified by* $\{<, +, \times, 0, 1\}$. *Here* $<$ *is a 2-place relation,* $+$ *and* $\times$ *are 2-place functions and* 0, 1 *are constants. Equality is a special 2-place relation that we will include in every language.*

We now describe how first-order sentences are built up from a given language $\mathscr{L}$.

Definition 1.3. *The set of terms in a language $\mathscr{L}$, denoted Term($\mathscr{L}$), is recursively defined by*

1. *each variable and constant is a term; and*
2. *if $t_1, \dots, t_n$ are terms and F is an n-place function symbol, then $F(t_1, \dots, t_n)$ is a term.*

A constant term is a term with no variables.

Definition 1.4. *Let $\mathscr{L}$ be a first-order language. The collection of $\mathscr{L}$-formulas is defined by recursion. First, the set of atomic formulas, denoted Atom($\mathscr{L}$), consists of formulas of one of the following forms:*

1. *$P(t_1, \dots, t_n)$ where P is an n-place predicate letter and $t_1, \dots, t_n$ are terms; and*
2. *$t_1 = t_2$ where t_1 and t_2 are terms.*

The set of $\mathscr{L}$-formulas is closed under the following rules

3. *If ϕ and θ are $\mathscr{L}$-formulas, then $(\phi \vee \theta)$ is an $\mathscr{L}$-formula. (Similarly, $(\phi \,\&\, \theta)$, $(\phi \to \theta)$, $(\phi \longleftrightarrow \theta)$, are $\mathscr{L}$-formulas.)*
4. *If ϕ is an $\mathscr{L}$-formula, then $(\neg\phi)$ is an $\mathscr{L}$-formula.*
5. *If ϕ is an $\mathscr{L}$-formula, then $(\exists v)\phi$ is an $\mathscr{L}$-formula (as is $(\forall v)\phi$).*

An example of an atomic formula in the language of arithmetic

$$0 + x = 0.$$

An example of a more complicated formula in the language, of plane geometry is the statement that every element either has a point incident with it or is incident with some line.

$$(\forall v)(\exists x)((xInv) \vee (vInx)).$$

A variable v that occurs in a formula ϕ becomes *bound* when it is placed in the scope of a quantifier, that is, $(\exists v)$ is placed in front of ϕ, and otherwise v is *free*. The concept of being *free* over-rides the concept of being *bound* in the sense that if a formula has both free and bound occurrences of a variable v, then v occurs free in that formula. The formal definition of bound and free variables is given by recursion.

Definition 1.5. *A variable v is free in a formula ϕ if*

1. *ϕ is atomic;*
2. *ϕ is $(\psi \vee \theta)$ and v is free in whichever one of ψ and θ in which it appears;*
3. *ϕ is $(\neg\psi)$ and v is free in ψ;*
4. *ϕ is $(\exists y)\psi$, v is free in ψ and y is not v.*

Example 1.6.

1. *In the atomic formula $x + 5 = 12$, the variable x is free.*
2. *In the formula $(\exists x)(x + 5 = 12)$, the variable x is bound.*
3. *In the formula $(\exists x)[(x \in \mathfrak{R}^+) \,\&\, (|x - 5| = 10)]$, the variable x is bound.*

We will refer to an $\mathscr{L}$-formula with no free variables as an *$\mathscr{L}$-sentence.*

2. Models and Interpretations

In propositional logic, we used truth tables and interpretations to consider the possible truth of complex statements in terms of their simplest components. In predicate logic, to consider the possible truth of complex statements that involve quantified variables, we need to introduce models with universes from which we can select the possible values for the variables.

Definition 2.1. *Suppose that $\mathscr{L}$ is a first-order language with*

(i) *predicate symbols* $P_1, P_2, \ldots,$
(ii) *function symbols* $F_1, F_2, \ldots,$ *and*
(iii) *constant symbols* $c_1, c_2, \ldots.$

Then an $\mathscr{L}$-structure $\mathfrak{A}$ *consists of*

(a) *a nonempty set A (called the* domain *or* universe *of* $\mathfrak{A}$*),*
(b) *a relation* $P_i^{\mathfrak{A}}$ *on A corresponding to each predicate symbol* P_i*,*
(c) *a function* $F_i^{\mathfrak{A}}$ *on A corresponding to each function symbol* F_i*, and*
(d) *a element* $c_i^{\mathfrak{A}} \in A$ *corresponding to each constant symbol* c_i*.*

Each relation $P_i^{\mathfrak{A}}$ requires the same number of places as P_i, so that $P_i^{\mathfrak{A}}$ is a subset of A^r for some fixed r (called the *arity* of R_i.) In addition, each function $F_i^{\mathfrak{A}}$ requires the same number of places as F_i, so that $F_i^{\mathfrak{A}} : A^r \rightarrow A$ for some fixed r (called the *arity* of R_j).

Definition 2.2. *Given a* $\mathscr{L}$*-structure* $\mathfrak{A}$*, an* interpretation I into $\mathfrak{A}$ *is a function I from the variables and constants of* $\mathscr{L}$ *into the universe A of* $\mathfrak{A}$ *that respects the interpretations of the symbols in* $\mathscr{L}$*. In particular, we have*

(i) *for each constant symbol* c_j*,* $I(c_j) = c_j^{\mathfrak{A}}$*,*
(ii) *for each function symbol* F_i*, if* F_i *has parity n and* $t_1, \ldots, t_n$ *are terms such that* $I(t_1), I(t_2), \ldots I(t_n)$ *have been defined, then*

$$I(F_i(t_1, \ldots, t_n)) = F_i^{\mathfrak{A}}(I(t_1), \ldots, I(t_n)).$$

For any interpretation I and any variable or constant x and for any element b of the universe, let $I_{b/x}$ be the interpretation defined by

$$I_{b/x}(z) = \begin{cases} b & \text{if } z = x, \\ I(z) & \text{otherwise.} \end{cases}$$

Definition 2.3. *We define by recursion the relation that a structure* $\mathfrak{A}$ *satisfies a formula* ϕ *via an interpretation I into* $\mathfrak{A}$*, denoted n* $\mathfrak{A} \models_I \phi$*:*

For atomic formulas, we have:

(1) $\mathfrak{A} \models_I t = s$ *if and only if* $I(t) = I(s)$*;*
(2) $\mathfrak{A} \models_I P_i(t_1, \ldots, t_n)$ *if and only if* $P_i^{\mathfrak{A}}(I(t_1), \ldots, I(t_n))$*.*

For formulas built up by the logical connectives we have:

(3) $\mathfrak{A} \models_I (\phi \vee \theta)$ *if and only if* $\mathfrak{A} \models_I \phi$ *or* $\mathfrak{A} \models_I \theta$*;*
(4) $\mathfrak{A} \models_I (\phi \ \& \ \theta)$ *if and only if* $\mathfrak{A} \models_I \phi$ *and* $\mathfrak{A} \models_I \theta$*;*
(5) $\mathfrak{A} \models_I (\phi \rightarrow \theta)$ *if and only if* $\mathfrak{A} \not\models_I \phi$ *or* $\mathfrak{A} \models_I \theta$*;*
(6) $\mathfrak{A} \models_I (\neg\phi)$ *if and only if* $\mathfrak{A} \not\models_I \phi$*.*

For formulas built up with quantifiers:

(7) $\mathfrak{A} \models_I (\exists v)\phi$ *if and only if there is an a in A such that* $\mathfrak{A} \models_{I_{a/x}} \phi$*;*
(8) $\mathfrak{A} \models_I (\forall v)\phi$ *if and only if for every a in A,* $\mathfrak{A} \models_{I_{a/x}} \phi$*.*

If $\mathfrak{A} \models_I \phi$ for every interpretation I, we will suppress the subscript I, and simply write $\mathfrak{A} \models \phi$. In this case we say that $\mathfrak{A}$ is a *model* of ϕ.

Example 2.4. *Let* $\mathscr{L}(GT)$ *be the language of group theory, which uses the symbols* $\{+, 0\}$*. A structure for this language is* $\mathfrak{A} = (\{0, 1, 2\}, +_{(\text{mod } 3)}, 0)$*. Suppose we consider formulas of* $\mathscr{L}(GT)$ *which only have variables among* x_1, x_2, x_3, x_4*. Define an interpretation I by* $I(x_i) \equiv i \mod 3$ *and* $I(0) = 0$*.*

(1) *Claim:* $\mathfrak{A} \not\models_I x_1 + x_2 = x_4$.
We check this claim by computation. Note that $I(x_1) = 1$, $I(x_2) = 2$, $I(x_1+x_2) = I(x_1) +_{mod 3} I(x_2) = 1 +_{mod 3} 2 = 0$. *On the other hand,* $I(x_4) = 1 \neq 0$, *so* $\mathfrak{A} \not\models_I x_1 + x_2 = x_4$.
(2) *Claim:* $\mathfrak{A} \models (\exists x_2)(x_1 + x_2 = x_4)$
Define $J = I_{0/x_2}$. *As above check that* $\mathfrak{A} \models_J x_1 + x_2 = x_4$. *Then by the definition of the satisfaction of an existential formula,* $\mathfrak{A} \models_I (\exists x_2)(x_1 + x_2 = x_4)$.

Theorem 2.5. *For every* $\mathscr{L}$*-formula* ϕ*, for all interpretations* I, J*, if* I *and* J *agree on all the variables free in* ϕ*, then* $\mathfrak{A} \models_I \phi$ *if and only if* $\mathfrak{A} \models_J \phi$.

Proof. Left to the reader. □

Corollary 2.6. *If* ϕ *is an* $\mathscr{L}$*-sentence, then for all interpretations* I *and* J*, we have* $\mathfrak{A} \models_I \phi$ *if and only if* $\mathfrak{A} \models_J \phi$.

Remark 2.7. *Thus for* $\mathscr{L}$*-sentences, we drop the subscript which indicates the interpretation of the variables, and we say simply* $\mathfrak{A}$ *models* ϕ.

Definition 2.8. *Let* ϕ *be an* $\mathscr{L}$*-formula.*
(i) ϕ *is logically valid if* $\mathfrak{A} \models_I \phi$ *for every* $\mathscr{L}$*-structure* $\mathfrak{A}$ *and every interpretation* I *into* $\mathfrak{A}$.
(ii) ϕ *is satisfiable if there is some* $\mathscr{L}$*-structure* $\mathfrak{A}$ *and some interpretation* I *into* $\mathfrak{A}$ *such that* $\mathfrak{A} \models_I \phi$.
(iii) ϕ *is contradictory if* ϕ *is not satisfiable.*

Definition 2.9. *A* $\mathscr{L}$*-theory* Γ *is a set of* $\mathscr{L}$*-sentences. An* $\mathscr{L}$*-structure* $\mathfrak{A}$ *is a model of an* $\mathscr{L}$*-theory* Γ *if and only if* $\mathfrak{A} \models \phi$ *for all* ϕ *in* Γ*. In this case we also say that* Γ *is satisfiable.*

Definition 2.10. *For a set of* $\mathscr{L}$*-formulas* Γ *and an* $\mathscr{L}$*-formula* ϕ*, we write* $\Gamma \models \phi$ *and say "*Γ *implies* ϕ*," if for all* $\mathscr{L}$*-structures* $\mathfrak{A}$ *and for all* $\mathscr{L}$*-interpretations* I*, if* $\mathfrak{A} \models_I \gamma$ *for all* γ *in* Γ*, then* $\mathfrak{A} \models_I \phi$.

Thus if Γ is an $\mathscr{L}$-theory and ϕ an $\mathscr{L}$-sentence, then $\Gamma \models \phi$ means every model of Γ is also a model of ϕ.

The following definition will be useful to us in the next section.

Definition 2.11. *Given a term* t *and an* $\mathscr{L}$*-formula* ϕ *with free variable* x*, we write* $\phi[t/x]$ *to indicate the result of substituting the term* t *for each free occurrence of* x *in* ϕ.

Example 2.12. *If* ϕ *is the formula* $(\exists y)(y \neq x)$ *is the formula, then* $\phi[y/x]$ *is the formula* $(\exists y)(y \neq y)$*, which we expect never to be true.*

3. The Deductive Calculus

The Predicate Calculus is a system of axioms and rules which permit us to derive the true statements of predicate logic without the use of interpretations. The basic relation in the Predicate Calculus is the relation *proves* between a set Γ of $\mathscr{L}$ formulas and an $\mathscr{L}$-formula ϕ, which formalizes the concept that Γ proves ϕ. This relation is denoted $\Gamma \vdash \phi$. As a first step in defining this relation, we give a list of additional rules of deduction, which extend the list we gave for propositional logic.

Some of our rules of the predicate calculus require that we exercise some care in how we substitute variables into certain formulas. Let us say that $\phi[t/x]$ is a *legal substitution* of t for x in ϕ if no free occurrence of x in ϕ occurs in the scope of a quantifier of any variable appearing in t. For instance, if ϕ has the form $(\forall y)\phi(x, y)$, where x is free, I cannot legally substitute y in for x, since then y would be bound by the universal quantifier.

10. (Equality rule) For any term t, the formula $t = t$ may be derived from Γ is one step.

11. (Term Substitution) For any terms $t_1, t_2, \ldots, t_n$, $s_1, s_2, \ldots, s_n$, and any function symbol F, if each of the sentences $t_1 = s_1, t_2 = s_2, \ldots, t_n = s_n$ have been derived from Γ, then we may derive $F(t_1, t_2, \ldots, t_n) = F(s_1, s_2, \ldots, s_n)$ from Γ in one additional step.

12. (Atomic Formula Substitution) For any terms $t_1, t_2, \ldots, t_n$, $s_1, s_2, \ldots, s_n$ and any atomic formula ϕ, if each of the sentences $t_1 = s_1, t_2 = s_2, \ldots, t_n = s_n$, and $\phi(t_1, t_2, \ldots, t_n)$, have been derived from Γ, then we may derive $\phi(s_1, s_2, \ldots, s_n)$ from Γ in one additional step.

13. ($\forall$-Elimination) For any term t, if $\phi[t/x]$ is a legal substitution and $(\forall x)\phi$ has been derived from Γ, then we may derive $\phi[t/x]$ from Γ in one additional step.

14. ($\exists$-Elimination) To show that $\Gamma \cup \{(\exists x)\phi(x)\} \vdash \theta$, it suffices to show $\Gamma \cup \{\phi(y)\}$, where y is a new variable that does not appear free in any formula in Γ nor in θ.

15. ($\forall$-Introduction) Suppose that y does not appear free in any formula in Γ, in any temporary assumption, nor in $(\forall x)\phi$. If $\phi[y/x]$ has been derived from Γ, then we may derive $(\forall x)\phi$ from Γ in one additional step.

16. ($\exists$-Introduction) If $\phi[t/x]$ is a legal substitution and $\phi[t/x]$ has been derived from Γ, then we may derive $(\exists x)\phi$ from Γ in one additional step.

We remark on three of the latter four rules. First, the reason for the restriction on substitution in $\forall$-Elimination is that we need to ensure that t does not contain any free variable that would be become bound when we substitute t for x in ϕ. For example, consider the formula $(\forall x)(\exists y)x < y$ in the language of arithmetic. Let ϕ be the formula $(\exists y)x < y$, in which x is free but y is bound. Observe that if we substitute the term y for x in ϕ, the resulting formula is $(\exists y)y < y$. Thus, from $(\forall x)(\exists y)x < y$ we can derive, for instance, $(\exists y)x < y$ or $(\exists y)c < y$, but we cannot derive $(\exists y)y < y$.

Second, the idea behind $\exists$-Elimination is this: Suppose in the course of my proof I have derived $(\exists x)\phi(x)$. Informally, I would like to use the fact that ϕ holds of some x, but to do so, I need to refer to this object. So I pick an unused variable, say a, and use this as a temporary name to stand for the object satisfying ϕ. Thus, I can write down $\phi(a)$. Eventually in my proof, I will discard this temporary name (usually by $\exists$-Introduction).

Third, in $\forall$-Introduction, if we think of the variable y as an arbitrary object, then when we show that y satisfies ϕ, we can conclude that ϕ holds of *every* object. However, if y is free in a premise in Γ or a temporary assumption, it is not arbitrary. For example, suppose we begin with the statement $(\exists x)(\forall z)(x + z = z)$ in the language of arithmetic and suppose we derive $(\forall z)(y + z = z)$ by $\exists$-Elimination (where y is a temporary name). We are *not* allowed to apply $\forall$-Introduction here, for otherwise we could conclude $(\forall x)(\forall z)(x + z = z)$, an undesirable conclusion.

Definition 3.1. *The relation $\Gamma \vdash \phi$ is the smallest subset of pairs (Γ, ϕ) from $\mathscr{P}(Sent) \times Sent$ that contains every pair (Γ, ϕ) such that $\phi \in \Gamma$ or ϕ is $t = t$ for some term t, and which is closed under the 15 rules of deduction.*

As in Propositional Calculus, to demonstrate that $\Gamma \vdash \phi$, we construct a proof. The next proposition exhibits several proofs using the new axiom and rules of predicate logic.

Proposition 3.2.

1. $(\exists x)(x = x)$.
2. $(\forall x)(\forall y)[x = y \rightarrow y = x]$.
3. $(\forall x)(\forall y)(\forall z)[(x = y \ \& \ y = z) \rightarrow x = z]$.
4. $((\forall x)\theta(x)) \rightarrow (\exists x)\theta(x)$.
5. $((\exists x)(\forall y)\theta(x, y)) \rightarrow (\forall y)(\exists x)\theta(x, y)$.
6. *(i)* $(\exists x)[\phi(x) \vee \psi(x)] \vdash (\exists x)\phi(x) \vee (\exists x)\psi(x)$
   (ii) $(\exists x)\phi(x) \vee (\exists x)\psi(x) \vdash (\exists x)[\phi(x) \vee \psi(x)]$
7. *(i)* $(\forall x)[\phi(x) \ \& \ \psi(x)] \vdash (\forall x)\phi(x) \ \& \ (\forall x)\psi(x)$
   (ii) $(\forall x)\phi(x) \ \& \ (\forall x)\psi(x) \vdash (\forall x)[\phi(x) \vee \psi(x)]$
8. $(\exists x)[\phi(x) \ \& \ \psi(x)] \vdash (\exists x)\phi(x) \ \& \ (\exists x)\psi(x)$
9. $(\forall x)\phi(x) \vee (\forall x)\psi(x) \vdash (\forall x)[\phi(x) \vee \psi(x)]$
10. $(\forall x)[\phi(x) \rightarrow \psi(f(x))] \rightarrow [(\exists x)\phi(x) \ \rightarrow \ (\exists x)\psi(x)]$.

Proof. 1. $(\exists x)(x = x)$

1	$x = x$	equality rule
2	$(\exists x)\, x = x$	$\exists$-Introduction 1

2. $(\forall x)(\forall y)[x = y \rightarrow y = x]$.

1	$\quad$ | $x = y$	temporary assumption
2	$\quad$ | $x = x$	equality rule
3	$\quad$ | $y = x$	term substitution 1,2
4	$x = y \rightarrow y = x$	$\rightarrow$-Introduction 1-3
5	$(\forall y)(x = y \rightarrow y = x)$	$\forall$-Introduction 4
6	$(\forall x)(\forall y)(x = y \rightarrow y = x)$	$\forall$-Introduction 5

5. $(\exists x)(\forall y)\theta(x, y) \vdash (\forall y)(\exists x)\theta(x, y)$.

1	$(\exists x)(\forall y)\theta(x, y)$	given rule
2	$(\forall y)\theta(a, y)$	$\exists$-Elimination 1
3	$\theta(a, y)$	$\forall$-Elimination 2
4	$(\exists x)\, \theta(x, y)$	$\exists$-Introduction 3
5	$(\forall y)(\exists x)\, \theta(x, y)$	$\forall$-Introduction 4

8. $(\exists x)[\phi(x) \,\&\, \psi(x)] \vdash (\exists x)\phi(x) \,\&\, (\exists x)\psi(x)$

1	$(\exists x)[\phi(x) \,\&\, \psi(x)]$	given rule
2	$\phi(a) \,\&\, \psi(a)$	$\exists$-Elimination 1
3	$\phi(a)$	&-Elimination 2
4	$(\exists x)\phi(x)$	$\exists$-Introduction 3
5	$\psi(a)$	&-Elimination 2
6	$(\exists x)\psi(x)$	$\exists$-Introduction 5
7	$(\exists x)\phi(x) \,\&\, (\exists x)\psi(x)$	&-Introduction 4,6

9. $(\forall x)\,\phi(x) \vee (\forall x)\,\psi(x) \vdash (\forall x)[\phi(x) \vee \psi(x)]$

1	$(\forall x)\,\phi(x) \vee (\forall x)\,\psi(x)$	given rule
2	$(\forall x)\,\phi(x)$	temporary assumption
3	$\phi(x)$	$\forall$-Elimination 2
4	$\phi(x) \vee \psi(x)$	$\vee$-Introduction 3
5	$(\forall x)[\phi(x) \vee \psi(x)]$	$\forall$-Introduction 4
6	$(\forall x)\,\psi(x)$	temporary assumption
7	$\psi(x)$	$\forall$-Elimination 6
8	$\phi(x) \vee \psi(x)$	$\vee$-Introduction 7
9	$(\forall x)[\phi(x) \vee \psi(x)]$	$\forall$-Introduction 8
10	$(\forall x)[\phi(x) \vee \psi(x)]$	$\vee$-Elimination 1-9

10. $(\forall x)[\phi(x) \to \psi(f(x))] \vdash (\exists x)\phi(x) \ \to\ (\exists x)\psi(x)$.

1	$(\forall x)[\phi(x) \to \psi(f(x))]$	given rule
2	$(\exists x)\phi(x)$	temporary assumption
3	$\phi(a)$	$\exists$-Elimination 2
4	$\phi(a) \to \psi(f(a))$	$\forall$-Elimination 1
5	$\psi(f(a))$	$\to$-Elimination 3,4
6	$(\exists x)\,\psi(x)$	$\exists$-Introduction 5
7	$(\exists x)\phi(x) \ \to\ (\exists x)\psi(x)$	$\to$-Introduction 2-6

□

4. Soundness Theorem for Predicate Logic

Our next goal is to proof the soundness theorem for predicate logic. First we will prove a lemma, which connects satisfaction of formulas with substituted variables to satisfaction with slightly modified interpretations of the original formulas.

Lemma 4.1. *For every $\mathscr{L}$-formula ϕ, every variable x, every term t, every structure $\mathfrak{B}$ and every interpretation I in $\mathfrak{B}$, if no free occurrence of x occurs in the scope of a quantifier over any variable appearing in t, then*

$$\mathfrak{B} \models_I \phi[t/x] \text{ if and only if } \mathfrak{B} \models_{I_{b/x}} \phi$$

where $b = I(t)$.

Proof. Let $\mathfrak{B} = (B, R_1, \dots, f_1, \dots, b_1, \dots)$ be an $\mathscr{L}$-structure, and let x, t, and I be as above. We claim that for any term r, if $b = I(t)$, then $I(r[t/x]) = I_{b/x}(r)$. We prove this claim by induction on the term r.

— If $r = a$ is a constant, then $r[t/x] = a$ so that $I(r[t/x]) = a^{\mathscr{B}} = I_{b/x}(r)$.
— If r is a variable $y \neq x$, then $r[t/x] = y$ and $I_{b/x}(y) = I(y)$, so that $I(r[t/x]) = I(y) = I_{b/x}(r)$.
— If $r = x$, then $r[t/x] = t$ and $I_{b/x}(x) = b$, so that $I(r[t/x]) = I(t) = b = I_{b/x}(r)$.
— Now assume the claim holds for terms $r_1, \dots, r_n$ and let $r = f(r_1, \dots, r_n)$ for some function symbol f. Then by induction $I(r_j[t/x]) = I_{b/x}(r_j)$ for $j = 1, 2, \dots, n$. Then

$$r[t/x] = f(r_1[t/x], \dots, r_n[t/x])$$

so

$$\begin{aligned} I_{b/x}(r) &= f^{\mathscr{B}}(I_{b/x}(r_1), \dots, I_{b/x}(r_n)) \\ &= f^{\mathscr{B}}(I(r_1[t/x]), \dots, I(r_n[t/x])) \\ &= I(f(r_1[t/x], \dots, r_n[t/x]) \\ &= I(r[t/x]). \end{aligned}$$

To prove the lemma, we proceed by induction on formulas.

— For an atomic formula ϕ of the form $s_1 = s_2$, we have

$$\begin{aligned}\mathscr{B} \models_I \phi[t/x] &\Leftrightarrow \mathscr{B} \models_I s_1[t/x] = s_2[t/x]\\ &\Leftrightarrow I(s_1[t/x]) = I(s_2[t/x])\\ &\Leftrightarrow I_{b/x}(s_1) = I_{b/x}(s_2) \text{ (by the claim)}\\ &\Leftrightarrow \mathscr{B} \models_{I_{b/x}} s_1 = s_2\\ &\Leftrightarrow \mathscr{B} \models_{I_{b/x}} \phi.\end{aligned}$$

— For an atomic formula ϕ of the form $P(r_1,\dots,r_n)$, so that $\phi[t/x]$ is $P(r_1[t/x],\dots,r_n[t/x])$, we have

$$\begin{aligned}\mathscr{B} \models_I \phi[t/x] &\Leftrightarrow \mathscr{B} \models_I P(r_1[t/x],\dots,r_n[t/x])\\ &\Leftrightarrow P^{\mathscr{B}}(I(r_1[t/x]),\dots,I(r_n[t/x]))\\ &\Leftrightarrow P^{\mathscr{B}}(I_{b/x}(r_1),\dots,I_{b/x}(r_n)) \text{ (by the claim)}\\ &\Leftrightarrow \mathscr{B} \models_{I_{b/x}} P(r_1,\dots,r_n)\\ &\Leftrightarrow \mathscr{B} \models_{I_{b/x}} \phi.\end{aligned}$$

— The inductive step for $\mathscr{L}$-formulas is straightforward except for formulas of the form $\forall y \phi$: Let ψ be $\forall y \phi$, where the Lemma holds for the formula ϕ. Then

$$\begin{aligned}\mathscr{B} \models_I \psi[t/x] &\Leftrightarrow \mathscr{B} \models_I \forall y \phi[t/x]\\ &\Leftrightarrow \mathscr{B} \models_{I_{a/y}} \phi[t/x] \text{ (for each } a \in B)\\ &\Leftrightarrow \mathscr{B} \models_{(I_{a/y})_{b/x}} \phi \text{ (by the inductive hypothesis)}\\ &\Leftrightarrow \mathscr{B} \models_{(I_{b/x})_{a/y}} \phi \text{ (for each } a \in B)\\ &\Leftrightarrow \mathscr{B} \models_{I_{b/x}} \forall y \phi\\ &\Leftrightarrow \mathscr{B} \models_{I_{b/x}} \psi.\end{aligned}$$

□

Theorem 4.2 (Soundness Theorem of Predicate Logic). *If* $\Gamma \vdash \phi$, *then* $\Gamma \models \phi$.

Proof. As in the proof of the soundness theorem for propositional logic, the proof is again by induction on the length of the deduction of ϕ. We need to show that if there is a proof of ϕ from Γ, then for any structure $\mathscr{A}$ and any interpretation I into $\mathscr{A}$, if $\mathscr{A} \models_I \gamma$ for all $\gamma \in \Gamma$, then $\mathscr{A} \models_I \phi$. The arguments for the rules from Propositional Logic carry over here, so we just need to verify the result holds for the new rules.

Suppose the result holds for all formulas obtained in proofs of length strictly less than n lines.

— (Equality rule) Suppose the last line of a proof of length n with premises Γ is $t = t$ for some term t. Suppose $\mathscr{A} \models_I \Gamma$. Then since $I(t) = I(t)$, we have $\mathscr{A} \models t = t$.

— (Term substitution) Suppose the last line of a proof of length n with premises Γ is $F(s_1,\dots,s_n) = F(t_1,\dots,t_n)$, obtained by term substitution. Then we must have established $s_1 = t_1,\dots,s_n = t_n$ earlier in the proof. By the inductive hypothesis, we must have $\Gamma \models s_1 = t_1, \dots \Gamma \models s_n = t_n$. Suppose that $\mathscr{A} \models_I \gamma$ for every $\gamma \in \Gamma$. Then $I(s_i) = I(t_i)$ for

$i = 1, \ldots n$. So

$$\begin{aligned} I(F(s_1, \ldots, s_n)) &= F^{\mathscr{A}}(I(s_1), \ldots, I(s_n)) \\ &= F^{\mathscr{A}}(I(t_1), \ldots, I(t_n)) \\ I(F(I(s_1), \ldots, I(s_n)) \end{aligned}$$

Hence $\mathscr{A} \models_I F(s_1, \ldots, s_n) = F(t_1, \ldots, t_n)$.

— (Atomic formula substitution) The argument is similar to the previous one and is left to the reader.

— ($\forall$-Elimination) *For any term t, if $\phi[t/x]$ is a legal substitution and $(\forall x)\phi$ has been derived from Γ, then we may derive $\phi[t/x]$ from Γ in one additional step.*

Suppose that the last line of a proof of length n with premises Γ is $\phi[t/x]$, obtained by $\forall$-Elimination. Thus, we must have derived $\forall x \phi(x)$ earlier in the proof. Let $\mathscr{A} \models_I \gamma$ for every $\gamma \in \Gamma$. Then by the inductive hypothesis, we have $\Gamma \models \forall x \phi(x)$, which implies that $\mathscr{A} \models_{I_{a/x}} \phi(x)$ for every $a \in A$. If $I(t) = b$, then since $\mathscr{A} \models_{I_{b/x}} \phi(x)$, by Lemma 4.1 we have $\mathscr{A} \models_I \phi[t/x]$. Since $\mathscr{A}$ and I were arbitrary, we can conclude that $\Gamma \models \phi[t/x]$.

— ($\exists$-Elimination) *To show that $\Gamma \cup \{(\exists x)\phi(x)\} \vdash \theta$, it suffices to show $\Gamma \cup \{\phi[y/x]\} \vdash \theta$, where y is a new variable that does not appear free in any formula in Γ nor in θ.*

Suppose that the last line of a proof of length n with premises Γ is given by $\exists$-Elimination. Then $\Gamma \vdash (\exists x)\phi(x)$ in less than n lines and $\Gamma \cup \{\phi[y/x]\} \vdash \theta$ in less than n lines. Let $\mathscr{A} \models_I \gamma$ for every $\gamma \in \Gamma$. Then by the inductive hypothesis, we have $\Gamma \models (\exists x)\phi(x)$, which implies that $\mathscr{A} \models_{I_{b/x}} \phi(x)$ for some $b \in A$. Let $J = I_{b/y}$, so that $J(y) = b$. It follows that $\mathscr{A} \models_{J_{b/x}} \phi(x)$, since $I = J$ except on possibly y and y does not appear free in ϕ. Then by Lemma 4.1, $\mathscr{A} \models_J \phi[y/x]$, and hence $\mathscr{A} \models_J \theta$. It follows that $\mathscr{A} \models_I \theta$, since $I = J$ except on possibly y and y does not appear free in θ. Since $\mathscr{A}$ and I were arbitrary, we can conclude that $\Gamma \cup (\exists x)\phi(x) \models \theta$.

— ($\forall$-Introduction) *Suppose that y does not appear free in any formula in Γ, in any temporary assumption, nor in $(\forall x)\phi$. If $\phi[y/x]$ has been derived from Γ, then we may derive $(\forall x)\phi$ from Γ in one additional step.*

Suppose that the last line of a proof of length n with premises Γ is $(\forall x)\phi(x)$, obtained by $\forall$-Introduction. Thus, we must have derived $\phi[y/x]$ from Γ earlier in the proof (where y satisfies the necessary conditions described above). Let $\mathscr{A} \models_I \gamma$ for every $\gamma \in \Gamma$. Since y does not appear free in Γ, then for any $a \in A$, $\mathscr{A} \models_{I_{a/y}} \Gamma$. For an arbitrary $a \in A$, let $J = I_{a/y}$, so that $J(y) = a$. By the inductive hypothesis, we have $\Gamma \models \phi[y/x]$, which implies that $\mathscr{A} \models_J \phi[y/x]$. Then by Lemma 4.1, $\mathscr{A} \models_{J_{a/x}} \phi$. Since $I_{a/x} = J_{a/x}$ except on possibly y, which does not appear free in ϕ, we have $\mathscr{A} \models_{I_{a/x}} \phi$. As a was arbitrary, we have shown $\mathscr{A} \models_{I_{a/x}} \phi$ for every $a \in A$. Hence $\mathscr{A} \models_I (\forall x)\phi(x)$. Since $\mathscr{A}$ and I were arbitrary, we can conclude that $\Gamma \models (\forall x)\phi(x)$.

— ($\exists$-Introduction) *If $\phi[t/x]$ is a legal substitution and $\phi[t/x]$ has been derived from Γ, then we may derive $(\exists x)\phi$ from Γ in one additional step.*

Suppose that the last line of a proof of length n with premises Γ is $(\exists x)\phi(x)$, obtained by $\exists$-Introduction. Thus, we must have derived $\phi[t/x]$ from Γ earlier in the proof, where t

is some term. Let $\mathscr{A} \models_I \gamma$ for every $\gamma \in \Gamma$. Then $I(t) = a$ for some $a \in A$. Since $\Gamma \vdash \phi[t/x]$ in less than n lines, by the inductive hypothesis, it follows that $\mathscr{A} \models_I \phi[t/x]$. Then by Lemma 4.1, $\mathscr{A} \models_{I_{a/x}} \phi$, which implies that $\mathscr{A} \models_I (\exists x)\phi(x)$. Since $\mathscr{A}$ and I were arbitrary, we can conclude that $\Gamma \models (\exists x)\phi(x)$.

□

FOUNDATIONS OF MATHEMATICS
CHAPTER 4: MODELS FOR PREDICATE LOGIC

In this chapter, we will prove the completeness theorem for predicate logic by showing how to build a model for a consistent first-order theory. We will also discuss several consequences of the compactness theorem for first-order logic and consider several relations that hold between various models of a given first-order theory, namely isomorphism and elementary equivalence.

1. The Completeness Theorem for Predicate Logic

Fix a first-order theory $\mathscr{L}$. For convenience, we will assume that our $\mathscr{L}$-formulas are built up only using $\neg, \vee$, and $\exists$. We will also make use of the following key facts (the proofs of which we will omit):

(1) If A is a tautology in propositional logic, then if we replace each instance of each propositional variable in A with an $\mathscr{L}$-formula, the resulting $\mathscr{L}$-formula is true in all $\mathscr{L}$-structures.
(2) For any $\mathscr{L}$-structure $\mathfrak{A}$ and any interpretation I into $\mathfrak{A}$,

$$\mathfrak{A} \models_I (\forall x)\phi \Leftrightarrow \mathfrak{A} \models_I \neg(\exists x)(\neg\phi).$$

We will also use the following analogues of results we proved in Chapter 2, the proofs of which are the same:

Lemma 1.1. *Let Γ be an $\mathscr{L}$-theory.*

(1) *If Γ is not consistent, then $\Gamma \vdash \phi$ for every $\mathscr{L}$-sentence ϕ.*
(2) *For an $\mathscr{L}$-sentence ϕ, $\Gamma \vdash \phi$ if and only if $\Gamma \cup \{\neg\phi\}$ is inconsistent.*
(3) *If Γ is consistent, then for any $\mathscr{L}$-sentence ϕ, either $\Gamma \cup \{\phi\}$ is consistent or $\Gamma \cup \{\neg\phi\}$ is consistent.*

The following result, known as the Constants Theorem, plays an important role in the proof of the completeness theorem.

Theorem 1.2 (Constants Theorem). *Let Γ be an $\mathscr{L}$-theory. If $\Gamma \vdash \phi(c)$ and c does not appear in Γ, then $\Gamma \vdash (\forall x)\phi(x)$.*

Proof. Given a proof of $\phi(c)$ from Γ, let v be a variable not appearing in Γ. If we replace every instance of c with v in the proof of $\phi(c)$, we have a proof of $\phi(v)$ from Γ. Then by $\forall$-Introduction, we have $\Gamma \vdash (\forall x)\phi(x)$. □

Gödel's completeness theorem can be articulated in two ways, which we will prove are equivalent:

Theorem 1.3 (Completeness theorem, Version 1). *For any $\mathscr{L}$-theory Γ and any $\mathscr{L}$-sentence ϕ,*

$$\Gamma \models \phi \Rightarrow \Gamma \vdash \phi.$$

Theorem 1.4 (Completeness theorem, Version 2). *Every consistent theory has a model.*

We claim that the two versions are equivalent.

Proof of claim. First, suppose that every consistent theory has a model, and suppose further that $\Gamma \models \phi$. If Γ is not consistent, then Γ proves every sentence, and hence $\Gamma \vdash \phi$. If, however, Γ is consistent, we have two cases to consider. If $\Gamma \cup \{\neg\phi\}$ is inconsistent, then by Lemma 1.1(2), it follows that $\Gamma \vdash \phi$. In the case that $\Gamma \cup \{\neg\phi\}$ is consistent, by the second version of the completeness theorem, there is some $\mathscr{L}$-structure $\mathfrak{A}$ such that $\mathfrak{A} \models \Gamma \cup \{\neg\phi\}$, from which it follows that $\mathfrak{A} \models \Gamma$ and $\mathfrak{A} \models \neg\phi$. But we have assumed that $\Gamma \models \phi$, and hence $\mathfrak{A} \models \phi$, which is impossible. Thus, if Γ is consistent, it follows that $\Gamma \cup \{\neg\phi\}$ is inconsistent.

For the other direction, suppose the first version of the completeness theorem holds and let Γ be an arbitrary $\mathscr{L}$-theory. Suppose Γ has no model. Then vacuously, $\Gamma \models \neg(\phi \vee \neg\phi)$, where ϕ is the sentence $(\forall x)x = x$. It follows from the first version of the completeness theorem that $\Gamma \vdash \neg(\phi \vee \neg\phi)$, and hence Γ is inconsistent. □

We now turn to the proof of the second version of completeness theorem. As in the proof of the completeness theorem for propositional logic, we will use the compactness theorem, which comes in several forms (just as it did in with propositional logic).

Theorem 1.5. *Let Γ be an $\mathscr{L}$-theory.*

(1) *For an $\mathscr{L}$-sentence ϕ, if $\Gamma \vdash \phi$, there is some finite $\Gamma_0 \subseteq \Gamma$, $\Gamma_0 \vdash \phi$.*
(2) *If every finite $\Gamma_0 \subseteq \Gamma$ is consistent, then Γ is consistent.*
(3) *If $\Gamma = \cup_n \Gamma_n$ is , $\Gamma_n \subseteq \Gamma_{n+1}$ for every n, and each Γ_n is consistent, then Γ is consistent.*

As in the case of propositional logic, (1) follows by induction on proof length, while (2) follows directly from (1) and (3) follows directly from (2).

Our strategy for proving the completeness theorem is as follows. Given Γ, we want to extend it to a maximally consistent collection of $\mathscr{L}$-formulas, like the proof of the completeness theorem for propositional logic. The problem that we now encounter (that did not occur in the propositional case) is that it is unclear how to make sentences of the form $(\exists x)\theta$.

The solution to this problem, due to Henkin, is to extend the language $\mathscr{L}$ to a language $\mathscr{L}'$ by adding new constants $c_0, c_1, c_2, \dots$, which we will use to witness the truth of existential sentences.

Hereafter, let us assume that $\mathscr{L}$ is countably infinite (which is not a necessary restriction), so that we will only need to add countably many new constants to our language. Using these constants, we will build a model of Γ, where the universe of our model consists of certain equivalence classes on the set of all $\mathscr{L}'$-terms with no variables (the so-called *Herbrand universe* of $\mathscr{L}'$). The model will satisfy a collection $\Delta \supseteq \Gamma$ that is maximally consistent and *Henkin complete*, which means that for each $\mathscr{L}'$-formula $\theta(v)$ with exactly one free variable v, if $(\exists v)\theta(v)$ is in Δ, then there is some constant c in our language such that $\theta(c)$ is in Δ.

Proof of Theorem 1.4. Let $\phi_0, \phi_1, \dots$ be an enumeration of all $\mathscr{L}'$-sentences. We define a sequence $\Gamma = \Delta_{-1} \subseteq \Delta_0 \subseteq \Delta_1 \subseteq \dots$ such that for each $n \in \mathbb{N}$,

$$\Delta_{2n} = \begin{cases} \Delta_{2n-1} \cup \{\phi_n\} & \text{if } \Delta_{2n-1} \cup \{\phi_n\} \text{ is consistent,} \\ \Delta_{2n-1} \cup \{\neg\phi_n\} & \text{otherwise} \end{cases}$$

and

$$\Delta_{2n+1} = \begin{cases} \Delta_{2n} \cup \{\theta(c_m)\} & \text{if } \phi_n \text{ is of the form } (\exists v)\theta(v) \text{ and is in } \Delta_{2n}, \\ \Delta_{2n} & \text{otherwise} \end{cases}$$

where c_m is the first constant in our list of new constants that has not appeared in Δ_{2n}. Then we define $\Delta = \cup_n \Delta_n$.

We now prove a series of claims.

Claim 1: Δ is complete (that is, for every $\mathscr{L}'$-sentence ϕ, either $\phi \in \Delta$ or $\neg\phi \in \Delta$).

Proof of Claim 1: This follows immediately from the construction.

Claim 2: Each Δ_k is consistent.

Proof of Claim 2: We prove this claim by induction. First, $\Delta_{-1} = \Gamma$ is consistent by assumption. Now suppose that Δ_k is consistent. If $k = 2n$ for some n, then clearly Δ_k is consistent, since if $\Delta_{2n-1} \cup \{\phi_n\}$ is consistent, then we set $\Delta_k = \Delta_{2n-1} \cup \{\phi_n\}$, and if not, then by Lemma 1.1(3), $\Delta_{2n-1} \cup \{\neg\phi_n\}$ is consistent, and so we set $\Delta_k = \Delta_{2n-1} \cup \{\neg\phi_n\}$.

If $k = 2n+1$ for some n, then if ϕ_n is not of the form $(\exists v)\theta(v)$ or if it is but it is not in Δ_{2n}, then $\Delta_{2n+1} = \Delta_{2n}$ is consistent by induction. If ϕ_n is of the form $(\exists v)\theta(v)$ and is in Δ_{2n}, then let $c = c_m$ be the first constant not appearing in Δ_{2n}. Suppose that $\Delta_k = \Delta_{2k+1} = \Delta_{2n} \cup \{\theta(c)\}$ is not consistent. Then by Lemma 1.1(2), $\Delta_{2n} \vdash \neg\theta(c)$. Then by the Constants Theorem, $\Delta_{2n} \vdash (\forall x)\neg\theta(x)$. But since ϕ_n is the formula $(\exists v)\theta(v)$ and is in Δ_{2n}, it follows that Δ_{2n} is inconsistent, contradicting our inductive hypothesis. Thus $\Delta_k = \Delta_{2n+1}$ is consistent.

Claim 3: $\Delta = \cup_n \Delta_n$ is consistent.

Proof of Claim 3: This follows from the third version of the compactness theorem.

Claim 4: Δ is Henkin complete (that is, for each $\mathscr{L}'$-formula $\theta(v)$ with exactly one free variable and $(\exists v)\theta(v) \in \Delta$, then $\theta(c) \in \Delta$ for some constant c).

Proof of Claim 4: Suppose that $(\exists v)\theta(v) \in \Delta$. Then there is some n such that $(\exists v)\theta(v)$ is the formula ϕ_n. Since $\Delta_{2n-1} \cup \{\phi_n\} \subseteq \Delta$ is consistent, $(\exists v)\theta(v) \in \Delta_{2n}$. Then by construction, $\theta(c) \in \Delta_{2n+1}$ for some constant c.

Our final task is to build a model $\mathfrak{A}$ such that $\mathfrak{A} \models \Delta$, from which it will follow that $\mathfrak{A} \models \Gamma$ (since $\Gamma \subseteq \Delta$). We define an equivalence relation on the Herbrand universe of $\mathscr{L}'$ (i.e., the set of constant $\mathscr{L}'$-terms, or equivalently, the $\mathscr{L}'$-terms that contain no variables). For constant terms s and t, we define

$$s \sim t \Leftrightarrow s = t \in \Delta.$$

Claim 5: $\sim$ is an equivalence relation.

Proof of Claim 5:

— Every sentence of the form $t = t$ must be in Δ since Δ is complete, so $\sim$ is reflexive.
— If $s = t \in \Delta$, then $t = s$ must also be in Δ since Δ is complete, so $\sim$ is symmetric.
— If $r = s, s = t \in \Delta$, then $r = t$ must also be in Δ since Δ is complete, so $\sim$ is transitive.

For a constant term s, let $[s]$ denote the equivalence class of s. Then we define an $\mathscr{L}'$-structure as follows:

(i) $A = \{[t] : t \text{ is a constant term of } \mathscr{L}'\}$;
(ii) for each function symbol f of the language $\mathscr{L}$, we define

$$f^{\mathfrak{A}}([t_1], \dots, [t_n]) = [f(t_1, \dots, t_n)],$$

where n is the arity of f;

(iii) for each predicate symbol P of the language $\mathscr{L}$, we define
$$P^{\mathfrak{A}}([t_1],\dots,[t_n]) \text{ if and only if } P(t_1,\dots,t_n)\in\Delta,$$
where n is the arity of P; and

(iv) for each constant symbol c of the language $\mathscr{L}'$, we define
$$c^{\mathfrak{A}} = [c].$$

Claim 6: $\mathfrak{A} = (A, f, \dots, P, \dots, c, \dots)$ is well-defined.

Proof of Claim 6: We have to show in particular that the interpretation of function symbols and predicate symbols in $\mathfrak{A}$ is well-defined. Suppose that $s_1 = t_1, \dots, s_n = t_n \in \Delta$ and
$$f^{\mathfrak{A}}([t_1],\dots,[t_n]) = [f(t_1,\dots,t_n)]. \tag{1}$$
By our first assumption, it follows that $\Delta \vdash s_i = t_i$ for $i = 1,\dots,n$. Then by term substitution, $\Delta \vdash f(s_1,\dots,s_n) = f(t_1,\dots,t_n)$, and so $f(s_1,\dots,s_n) = f(t_1,\dots,t_n)\in\Delta$. It follows that
$$[f(s_1,\dots,s_n)]. = [f(t_1,\dots,t_n)]. \tag{2}$$
Combining (1) and (2) yields
$$f^{\mathfrak{A}}([t_1],\dots,[t_n]) = [f(t_1,\dots,t_n)] = [f(s_1,\dots,s_n)] = f^{\mathfrak{A}}([s_1],\dots,[s_n]).$$
A similar argument shows that the interpretation of predicate symbols is well-defined.

Claim 7: Let I be an interpretation into $\mathfrak{A}$. Then $I(t) = [t]$ for every constant term t.

Proof of Claim 7: We verify this inductively for constant symbols and then for function symbols applied to constant terms.

— Suppose t is a constant symbol c. Then $I(c) = c^{\mathfrak{A}} = [c]$.

— Suppose that t is the term $f(t_1,\dots,t_n)$ for constant symbol f and constant terms $t_1,\dots,t_n$, where $I(t_i) = [t_i]$ for $i = 1,\dots,n$. Then
$$I(f(t_1,\dots,t_n)) = f^{\mathfrak{A}}(I(t_1),\dots,I(t_n)) = f^{\mathfrak{A}}([t_1],\dots,[t_n]) = [f(t_1,\dots,t_n))].$$

Claim 8: $\mathfrak{A} \models \Delta$. We verify this by proving that for every interpretation I into $\mathfrak{A}$ and every $\mathscr{L}'$-sentence ϕ, $\mathfrak{A} \models_I \phi$ if and only if $\phi\in\Delta$.

— If ϕ is $s = t$ for some terms s, t, then
$$\begin{aligned}\mathfrak{A} \models_I s = t &\Leftrightarrow I(s) = I(t)\\ &\Leftrightarrow [s] = [t]\\ &\Leftrightarrow s \sim t\\ &\Leftrightarrow s = t \in \Delta.\end{aligned}$$

— If ϕ is $P(t_1,\dots,t_n)$ for some predicate symbol P, then
$$\begin{aligned}\mathfrak{A} \models_I P(t_1,\dots,t_n) &\Leftrightarrow P^{\mathfrak{A}}(I(t_1),\dots,I(t_n))\\ &\Leftrightarrow P^{\mathfrak{A}}([t_1],\dots,[t_n])\\ &\Leftrightarrow P(t_1,\dots,t_n)\in\Delta.\end{aligned}$$

— If ϕ is $\neg\psi$ for some $\mathscr{L}'$-sentence ψ, then

$$\begin{aligned}\mathfrak{A} \models_I \neg\psi &\Leftrightarrow \mathfrak{A} \not\models \psi \\ &\Leftrightarrow \psi \notin \Delta \\ &\Leftrightarrow \neg\psi \in \Delta.\end{aligned}$$

— If ϕ is $\psi \vee \theta$ for some $\mathscr{L}'$-sentences ψ and θ, then

$$\begin{aligned}\mathfrak{A} \models_I \psi \vee \theta &\Leftrightarrow \mathfrak{A} \models \psi \text{ or } \mathfrak{A} \models \theta \\ &\Leftrightarrow \psi \in \Delta \text{ or } \theta \in \Delta \\ &\Leftrightarrow \psi \vee \theta \in \Delta.\end{aligned}$$

— If ϕ is $(\exists v)\theta(v)$ for some $\mathscr{L}'$-formula θ with one free variable v, then

$$\begin{aligned}\mathfrak{A} \models_I (\exists v)\theta(v) &\Leftrightarrow \mathfrak{A} \models_{I_{b/v}} \theta(v) \text{ for some } b \in A \\ &\Leftrightarrow \mathfrak{A} \models_I \theta(c) \text{ where } b = [c] \\ &\Leftrightarrow \theta(c) \in \Delta \\ &\Leftrightarrow (\exists v)\theta(v) \in \Delta.\end{aligned}$$

Since $\mathfrak{A} \models \Delta$, it follows that $\mathfrak{A} \models \Gamma$. Note that $\mathfrak{A}$ is an $\mathscr{L}'$-structure while Γ is only an $\mathscr{L}$-theory (as it does not contain any expression involving any of the additional constants). Then let $\mathfrak{A}^*$ be the $\mathscr{L}$-structure with the same universe as $\mathfrak{A}$ and the same interpretations of the function symbols and predicate symbols, but without interpreting the constants symbols that are in $\mathscr{L}' - \mathscr{L}$ (the so-called *reduct* of $\mathfrak{A}$). Then clearly $\mathfrak{A}^* \models \Gamma$, and the proof is complete.

□

2. Consequences of the completeness theorem

The same consequences we derived from the Soundness and Completeness Theorem for Propositional Logic apply now to Predicate Logic with basically the same proofs.

Theorem 2.1. *For any set of sentences Γ, Γ is satisfiable if and only if Γ is consistent.*

Theorem 2.2. *If Σ is a consistent theory, then Σ is included in some complete, consistent theory.*

We also have an additional version of the compactness theorem, which is the most common formulation of compactness.

Theorem 2.3 (Compactness Theorem for Predicate Logic).
An $\mathscr{L}$-theory Γ is satisfiable if and only if every finite subset of Γ is satisfiable.

Proof. ($\Rightarrow$) If $\mathfrak{A} \models \Gamma$, then it immediately follows that $\mathfrak{A} \models \Gamma_0$ for any finite $\Gamma_0 \subseteq \Gamma$.

($\Leftarrow$) Suppose that Γ is not satisfiable. By the completeness theorem, Γ is not consistent. Then $\Gamma \vdash \phi \,\&\, \neg\phi$ for some $\mathscr{L}$-sentence ϕ. Then by the first formulation of the compactness theorem there is some finite $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \vdash \phi \,\&\, \neg\phi$. It follows that Γ_0 is not satisfiable. □

We now consider two applications of the compactness theorem, the first yielding a model of arithmetic with infinite natural numbers and the second yielding a model of the real numbers with infinitesimals.

Example 2.4. *Let $\mathscr{L} = \{+, \times, <, 0, 1\}$ be the language of arithmetic, and let $\Gamma = Th(\mathbb{N})$, the set of $\mathscr{L}$-sentences true in the standard model of arithmetic. Let us expand $\mathscr{L}$ to $\mathscr{L}'$ by adding a new constant c to our language. We extend Γ to an $\mathscr{L}'$-theory Γ' by adding all sentences of the form*

$$\psi_n : c > \underbrace{1 + \ldots + 1}_{n \text{ times}}$$

We claim that every finite $\Gamma_0' \subseteq \Gamma'$ is satisfiable. Given any finite $\Gamma_0' \subseteq \Gamma'$, Γ_0' consists of at most finitely many sentences from Γ and at most finitely many sentences of the form ψ_i. It follows that

$$\Gamma_0' \subseteq \Gamma \cup \{\psi_{n_1}, \psi_{n_2}, \ldots, \psi_{n_k}\}$$

for some $n_1, n_2, \ldots, n_k \in \mathbb{N}$, where these latter sentences assert that c is larger than each of the values $n_1, n_2, \ldots, n_k$. Let $n = \max\{n_1, \ldots, n_k\}$ then let $\mathfrak{A} = (\mathbb{N}, +, \times, <, 0, 1, n)$, so that $c^{\mathfrak{A}} = n$ and hence $\mathfrak{A} \models \Gamma_0'$. Then by the compactness theorem, there is some $\mathscr{L}'$-structure $\mathfrak{B}$ such that $\mathfrak{B} \models \Gamma'$. In the universe of $\mathfrak{B}$, we have objects that behave exactly like $0, 1, 2, 3, \ldots$ (in a sense we will make precise shortly), but the interpretation of c in $\mathfrak{B}$ satisfies $c^{\mathfrak{B}} > n$ for every $n \in \mathbb{N}$ and hence behaves like an infinite natural number. We will write the universe of $\mathfrak{B}$ as $\mathbb{N}^*$.

Example 2.5. *Let $\mathscr{L}$ consist of*

— *an n-ary function symbol F_f for every $f : \mathbb{R}^n \to \mathbb{R}$;*
— *an n-ary predicate symbol P_A for every $A \subseteq \mathbb{R}^n$; and*
— *a constant symbol c_r for every $r \in \mathbb{R}$.*

Let $\mathfrak{R}$ be the $\mathscr{L}$-structure with universe $\mathbb{R}$ satisfying

— *$F_f^{\mathfrak{R}} = f$ for every function symbol F_f;*
— *$P_A^{\mathfrak{R}} = A$ for every predicate symbol P_A; and*
— *$c_r^{\mathfrak{R}} = r$ for every constant symbol c_r.*

Let us expand $\mathscr{L}$ to $\mathscr{L}'$ by adding a new constant d to our language. We extend Γ to an $\mathscr{L}'$-theory Γ' by adding all sentences of the form

$$\theta_r : c_0 < d < c_r$$

for $r \in \mathbb{R}^{>0}$. As in the previous example, every finite $\Gamma_0' \subseteq \Gamma'$ is satisfiable. Hence by the compactness theorem, Γ' is satisfiable. Let $\mathfrak{A} \models \Gamma$. The universe of $\mathfrak{A}$ contains a copy of $\mathbb{R}$ (in a sense we will make precise shortly). In addition, $d^{\mathfrak{A}}$ is infinitesimal object. For every real number in $\mathfrak{A}$, $0 < d^{\mathfrak{A}} < r$ holds. We will write the universe of $\mathfrak{A}$ as $\mathbb{R}^*$.

Now we consider a question that was not appropriate to consider in the context of propositional logic, namely, what are the sizes of models of a given theory? Our main theorem is a consequence of the proof of the Completeness Theorem. We proved the Completeness Theorem only in the case of a countable language L, and we built a countable model (which was possibly finite). By using a little care (and some set theory), one can modify steps (1) and (2) for an uncountable language to define by transfinite recursion a theory Δ and prove by transfinite induction that Δ has the desired properties. The construction leads to a model whose size is at most the size of the language with which one started. Thus we have:

Theorem 2.6 (Löwenheim-Skolem Theorem)**.** *If Γ is an $\mathscr{L}$-theory with an infinite model, then Γ has a model of size κ for every infinite κ with $|\mathscr{L}| \leq \kappa$.*

Proof Sketch. First we add κ new constant symbols $\langle d_\alpha \mid \alpha < \kappa \rangle$ to our language $\mathscr{L}$. Next we expand Γ to Γ' by adding formulas that say $d_\alpha \neq d_\beta$ for the different constants:

$$\Gamma' = \Gamma \cup \left\{ \neg d_\alpha = d_\beta : \alpha < \beta < \kappa \right\}.$$

Since Γ has an infinite model, each finite $\Gamma_0' \subseteq \Gamma'$ has a model. Hence by the compactness theorem, Γ' has a model. By the soundness theorem, Γ' is consistent. Then use the proof of the completeness theorem to define a model $\mathfrak{B}'$ of Γ' the universe of which has size $|B| \leq \kappa$. Since $\mathfrak{B}' \models d_\alpha \neq d_\beta$ for $\alpha \neq \beta$, there are at least κ many elements. Thus $|B| = \kappa$ and so Γ' has a model $\mathfrak{B}'$ of the desired cardinality. Let $\mathfrak{B}$ be the reduct of $\mathfrak{B}'$ obtained by removing the new constant symbols from our expanded language. Then $\mathfrak{B}$ is a model of the desired size for Γ. □

3. Isomorphism and elementary equivalence

We conclude this chapter with a discussion of isomorphic models and the notion of elementary equivalence.

Definition 3.1. *Given $\mathscr{L}$-structures $\mathfrak{A}$ and $\mathfrak{B}$, a bijection $H : \mathfrak{A} \to \mathfrak{B}$ is an isomorphism if it satisfies:*

(1) *For every constant $c \in \mathscr{L}$, $H(c^{\mathfrak{A}}) = c^{\mathfrak{B}}$.*

(2) *For every k-ary predicate symbol $P \in \mathscr{L}$ and every $a_1, \dots, a_k \in A$,*
$$P^{\mathfrak{A}}(a_1,\dots,a_k) \Leftrightarrow P^{\mathfrak{B}}(H(a_1),\dots,H(a_k)).$$

(3) *For every k-ary function symbol $F \in \mathscr{L}$ and every $a_1, \dots, a_k \in A$,*
$$H(F^{\mathfrak{A}}(a_1,\dots,a_k)) = F^{\mathfrak{B}}(H(a_1),\dots,H(a_k)).$$

Furthermore, $\mathfrak{A}$ and $\mathfrak{B}$ are isomorphic, denoted $\mathfrak{A} \cong \mathfrak{B}$, if there exists an isomorphism between $\mathfrak{A}$ and $\mathfrak{B}$.

Example 3.2. *The ordered group $(\mathbb{R}, +, <)$ of real numbers under addition is isomorphic to the ordered group $(\mathbb{R}^{>0}, \cdot, <)$ of positive real numbers under multiplication under the mapping $H(x) = 2^x$. The key observation here is that $H(x + y) = 2^{x+y} = 2^x \cdot 2^y = H(x) \cdot H(y)$.*

We compare the relation of isomorphism with the following relation between models.

Definition 3.3. *$\mathscr{L}$-structures $\mathfrak{A}$ and $\mathfrak{B}$ are* elementarily equivalent*, denoted $\mathfrak{A} \equiv \mathfrak{B}$, if for any $\mathscr{L}$-sentence ϕ,*
$$\mathfrak{A} \models \phi \Leftrightarrow \mathfrak{B} \models \phi.$$

How do the relations of $\cong$ and $\equiv$ compare? First, we have the following theorem.

Theorem 3.4. *If $\mathfrak{A}$ and $\mathfrak{B}$ are $\mathscr{L}$-structures satisfying $\mathfrak{A} \cong \mathfrak{B}$, then $\mathfrak{A} \equiv \mathfrak{B}$.*

The proof is by induction on the complexity of $\mathscr{L}$-sentences. The converse of this theorem does not hold, as shown by the following example.

Example 3.5. $(\mathbb{Q}, \leq)$ *and* $(\mathbb{R}, \leq)$*, both models of the theory of dense linear orders without endpoints, are elementarily equivalent, which follows from the fact that the theory of dense linear orders without endpoints is complete (which we will prove in Chapter 6). Note, however, that these structures are not isomorphic, since they have different cardinalities.*

We conclude this chapter with one last set of definitions and examples.

Definition 3.6. *Let $\mathfrak{A}$ and $\mathfrak{B}$ be $\mathscr{L}$-structures with corresponding domains $A \subseteq B$.*

(1) *$\mathfrak{A}$ is a* submodel *of $\mathfrak{B}$ ($\mathfrak{A} \subseteq \mathfrak{B}$) if the following are satisfied:*
   (a) *for each constant $c \in \mathscr{L}$, $c^{\mathfrak{A}} = c^{\mathfrak{B}}$;*
   (b) *for each n-ary function symbol $f \in \mathscr{L}$ and each $a_1, \dots, a_n \in A$,*
$$f^{\mathfrak{A}}(a_1,\dots,a_n) = f^{\mathfrak{B}}(a_1,\dots,a_n);$$

(c) *for each n-ary relation symbol* $R \in \mathscr{L}$ *and each* $a_1, \dots, a_n \in A$,

$$R^{\mathfrak{A}}(a_1, \dots, a_n) \Leftrightarrow R^{\mathfrak{B}}(a_1, \dots, a_n).$$

(2) $\mathfrak{A}$ *is an* elementary submodel *of* $\mathfrak{B}$ *(written* $\mathfrak{A} \precsim \mathfrak{B}$*) if*

(a) $\mathfrak{A}$ *is a submodel of* $\mathfrak{B}$*;*

(b) *for each* $\mathscr{L}$*-formula* $\phi(x_1, \dots, x_n)$ *and each* $a_1, \dots, a_n \in A$,

$$\mathfrak{A} \models \phi(a_1, \dots, a_n) \Leftrightarrow \mathfrak{B} \models \phi(a_1, \dots, a_n).$$

Example 3.7. *Consider the rings* $(\mathbb{Z}, 0, 1, +, \cdot) \subseteq (\mathbb{Q}, 0, 1, +, \cdot) \subseteq (\mathbb{R}, 0, 1, +, \cdot)$.

— $(\mathbb{Z}, 0, 1, +, \cdot)$ *is a submodel of* $(\mathbb{Q}, 0, 1, +, \cdot)$ *and* $(\mathbb{Q}, 0, 1, +, \cdot)$ *is a submodel of* $(\mathbb{R}, 0, 1, +, \cdot)$.

— $(\mathbb{Z}, 0, 1, +, \cdot)$ *is not an elementary submodel of* $(\mathbb{Q}, 0, 1, +, \cdot)$*, since* $\mathbb{Q} \models (\exists x) x + x = 1$ *which is false in* $\mathbb{Z}$.

— *Neither* $(\mathbb{Z}, 0, 1, +, \cdot)$ *nor* $(\mathbb{Q}, 0, 1, +, \cdot)$ *is an elementary submodel of* $(\mathbb{R}, 0, 1, +, \cdot)$ *since* $\mathbb{R} \models (\exists x) x \cdot x = 2$*, which is false in both* $\mathbb{Z}$ *and* $\mathbb{Q}$.

Example 3.8. *The following elementary submodel relations hold:*

— $(\mathbb{Q}, \leq) \precsim (\mathbb{R}, \leq)$

— $(\mathbb{N}, 0, 1, +, \cdot) \precsim (\mathbb{N}^*, 0, 1, +, \cdot)$.

— $(\mathbb{R}, 0, 1, +, \cdot) \precsim (\mathbb{R}^*, 0, 1, +, \cdot)$.

The latter two items in the previous example justify the claims that the natural numbers are contained in models of non-standard arithmetic and that the real numbers are contained in models of non-standard analysis.

We conclude with one last example.

Example 3.9. $\mathbb{Z}_3 = \{0, 1, 2\}$ *with addition modulo 3 is not a submodel of* $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$ *with addition modulo 6 because they have different addition functions: In* $\mathbb{Z}_3$*,* $2 + 2 = 1$ *whereas in* $\mathbb{Z}_6$*,* $2 + 2 = 4$*. However,* $\mathbb{Z}_3$ is *isomorphic to the subgroup of* $\mathbb{Z}_6$ *consisting of* $\{0, 2, 4\}$.

FOUNDATIONS OF MATHEMATICS
CHAPTER 5: COMPUTABILITY THEORY

1. Introduction and Examples

There are many different approaches to defining the collection of computable number-theoretic functions. The intuitive idea is that a function $F : \mathbb{N} \to \mathbb{N}$ (or more generally $F : \mathbb{N}^k \to \mathbb{N}$) is computable if there is an algorithm or effective procedure for determining the output $F(m)$ from the input m. To demonstrate that a particular function F is computable, it suffices to give the corresponding algorithm.

Example 1.1. Basic computable functions include

(i) the successor function $S(x) = x + 1$,
(ii) the addition function $+(x, y) = x + y$, and
(iii) the multiplication function $\cdot(x, y) = x \cdot y$.

Example 1.2. Some slightly more complicated examples of computable functions:

(i) The Division Algorithm demonstrates that the two functions that compute, for inputs a and b, the unique quotient $q = q(a, b)$ and remainder $r = r(a, b)$, with $0 \leq r < a$, such that $b = qa + r$, are both computable.
(ii) The Euclidean Algorithm demonstrates that the function $gcd(a, b)$ which computes the greatest common divisor of a and b is computable. It follows that least common multiple function $lcm(a, b) = (a \cdot b)/gcd(a, b)$ is also computable

The notion of computability for functions can be extended to subsets of $\mathbb{N}$ or relations on $\mathbb{N}^k$ for some k as follows. First, a set $A \subseteq \mathbb{N}$ is said to be computable if the characteristic function of A, defined by

$$\chi_A(n) = \begin{cases} 1 & \text{if } n \in A \\ 0 & \text{if } n \notin A, \end{cases}$$

is a computable function. Similarly, a relation $R \subseteq \mathbb{N}^k$ is said to be computable if its characteristic function

$$\chi_A(n_1, \dots, n_k) = \begin{cases} 1 & \text{if } (n_1, \dots, n_k) \in R \\ 0 & \text{if } (n_1, \dots, n_k) \notin R \end{cases}$$

is computable. These definitions are equivalent to saying that there is an algorithm for testing whether a given number is in A or whether a given finite sequence $(n_1, \dots, n_k)$ is in R.

Example 1.3. The following are computable:

(i) The set of perfect squares is computable, since given a number n, we can test whether it is a square by computing m^2 for all $m \leq n$ and checking whether $m^2 = n$.
(ii) The relation $x \mid y$ ("x divides y") is computable, since by the Division Algorithm, $x \mid y$ if and only if the remainder $r(x, y) = 0$.
(iii) The set of even numbers is computable, since n is even if and only if $2 \mid n$.

(iv) The set of prime numbers is computable, since p is prime if and only if

$$(\forall m < p)[(m \neq 0 \,\&\, m \neq 1) \rightarrow m \not| p].$$

Two formalizations of the class of computable functions that will we consider are the collection of Turing machines and the collection of partial recursive functions.

The first general model of computation that we will consider is the Turing machine, developed by Alan Turing in the 1930's. The machine consists of one or more infinite *tapes* with cells on which symbols from a finite alphabet may be written, together with *heads* which can read the contents of a given cell, write a new symbol on the cell, and move to an adjacent cell. A program for such a machine is given by a finite set of *states* and a transition function which describes the action taken in a given state when a certain symbol is scanned. Possible actions are (1) writing a new symbol in the cell; (2) moving to an adjacent cell; (3) switching to a new state.

2. Finite State Automata

As a warm-up, we will first consider a simplified version of a Turing machine, known as a finite state automaton. A finite state automaton over a finite alphabet Σ (usually $\{0,1\}$) is given by a finite set of states $Q = \{q_0, q_1, \dots, q_k\}$ and a transition function $\delta : Q \times \Sigma \rightarrow Q$. There may also be an output function $F : Q \times \Sigma \rightarrow \Sigma$. The state q_0 is designated as the *initial* state and there may be a set $A \subseteq Q$ of *accepting* states.

The action of a finite automaton M on input $w = a_0 a_1 \dots a_k$ occurs in *stages*.

— At stage 0, the machine begins in state q_0, scans the input a_0, and then and transitions to state $s_1 = \delta(q_0, a_0)$, possibly writing $b_0 = F(q_0, a_0)$.

— At stage n, the machine (in state s_n) scans a_n, transitions to state $s_{n+1} = F(s_n, a_n)$, and possibly writes $b_n = F(s_n, a_n)$.

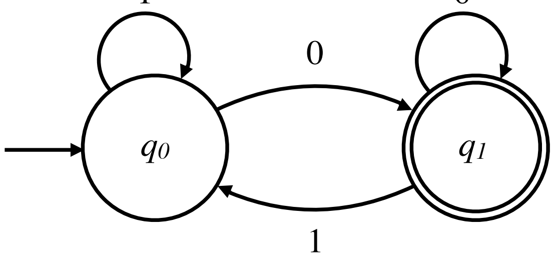
— After reading a_k during stage k, the machine halts in state s_{k+1}. The input word w is *accepted* by M if $s_{k+1} \in A$. If there is an ouput function, then the output word will be written as $M(w) = b_0 b_1 \dots b_k$.

The language $L(M)$ is the set of words accepted by M. We will sometimes refer to such a collection as a *regular language*.

Example 2.1. Let M_1 be the machine, depicted by the state diagram below, with transition function

$$\delta(q_i, j) = \begin{cases} q_{1-i} & \text{if } i = j \\ q_i & \text{if } i \neq j, \end{cases}$$

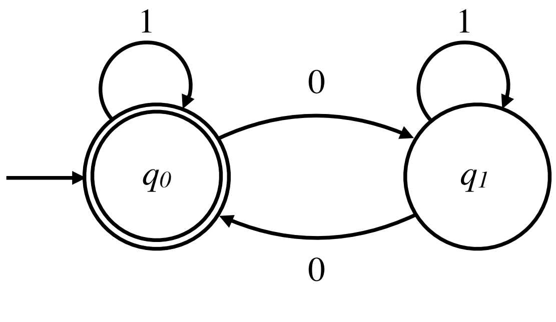
for $i = 0, 1$ and $j = 0, 1$. Thus $L(M_1)$ is the set of words which end in a 0.



Example 2.2. Let M_2 be the machine, depicted by the state diagram below, with transition function

$$\delta(q_i, j) = \begin{cases} q_{1-i} & \text{if } j = 0 \\ q_i & \text{if } j = 1, \end{cases}$$

for $i = 0, 1$. Thus $L(M_2)$ is the set of words which contain an even number of 0's.

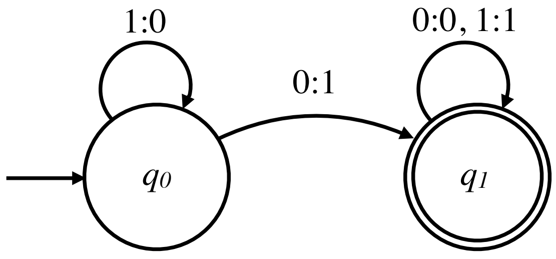


Next we consider some finite state automata which compute functions. We want to express natural numbers in reverse binary notation, so that the word $a_0 a_1 \dots a_k$ represents $a_0 + 2a_1 + \cdots + 2^k a_k$.

Example 2.3. The successor machine M_3 computes $S(x) = x + 1$. State q_0 is the *carry* state and state q_1 is the *no-carry* state. The edges in the state diagram below are labelled with symbols of the form $i : j$, which means that i is an input bit and j is an output bit.

For reasons that will be clear shortly, we require that any number of the form $2^n - 1$ (normally represented by a string of the form 1^n) to be represented by $1^n 0$. For any other number, adding additional zeros to the end of its representation will make no difference.

Starting in state q_0, the machine outputs $(i + 1) \mod 2$ on input i and transitions to state q_{1-i}. From state q_1 on input i, the machine outputs i and remains in state q_1. We take the liberty of adding an extra 0 at the end of the input in order to accommodate the carry.



More precisely, the transition function and output function of M_3 are defined by

$$\delta(q_i, j) = \begin{cases} q_0 & \text{if } i = 0 \;\&\; j = 1 \\ q_1 & \text{if } (i = 0 \;\&\; j = 0) \vee i = 1 \end{cases}$$

and

$$F(q_i, j) = \begin{cases} 1 - j & \text{if } i = 0 \\ j & \text{if } i = 1 \end{cases}.$$

We see that this computes $S(x)$ by the following reasoning. Assume that x ends with 0 (if x represents the number $2^n - 1$ for some n, then this terminal 0 is necessary to end up in an accepting state; if x does not, then the terminal 0 is inconsequential).

We consider two cases:

— Case 1: x contains a 0. Let i be least such that $a_i = 0$. Then M will write $i - 1$ 1's and remain in state q_0 until arriving at the i-th bit, which is a 0. Then it will output $b_i = 1$ and transition to state q_1. After that it will simply write $b_j = a_j$ for all $j > i$.

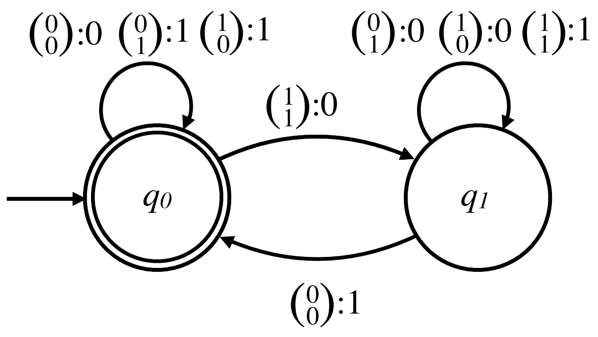
— Case 2: If x consists of all 1's, then M will write n 0's and remain in state q_0 until reaching the extra 0, when M will output a final 1 and transition to q_1.

In each case, $b_1 \ldots b_n = M(x)$ is the reverse binary representation of $S(x)$ and we will end up in an accepting state.

Using the carry and no-carry states, we can also perform addition with a finite state automaton.

Example 2.4. The addition machine M_4 computes $S(x, y) = x + y$. Unlike the previous example, state q_0 is the *no-carry* state and state q_1 is the *carry* state. Moreover, we work with a different input alphabet: the input alphabet consists of pairs $\begin{pmatrix} i \\ j \end{pmatrix} \in \{0,1\} \times \{0,1\}$. To add two numbers n_1 and n_2 represented by σ_1 and σ_2, if σ_1 is shorter than σ_2, we append 0s to σ_1 so that the resulting string has the same length as σ_2. As in the previous example, we will also append an additional 0 to the end of σ_1 and σ_2, which is necessary in case that the string representing $n_1 + n_2$ is strictly longer than the strings representing n_1 and n_2.

The state diagram is given by:


$\binom{0}{0}$:0 $\binom{0}{1}$:1 $\binom{1}{0}$:1
$\binom{0}{1}$:0 $\binom{1}{0}$:0 $\binom{1}{1}$:1
$\binom{1}{1}$:0
q_0
q_1
$\binom{0}{0}$:1

The transition function and output function of M_3 are defined in the following tables:

δ	$\begin{pmatrix}0\\0\end{pmatrix}$	$\begin{pmatrix}0\\1\end{pmatrix}$	$\begin{pmatrix}1\\0\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$
q_0	q_0	q_0	q_0	q_1
q_1	q_0	q_1	q_1	q_1

F	$\begin{pmatrix}0\\0\end{pmatrix}$	$\begin{pmatrix}0\\1\end{pmatrix}$	$\begin{pmatrix}1\\0\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$
q_0	0	1	1	0
q_1	1	0	0	1

For any finite state automaton M, the set $L(M)$ of words accepted by M is a computable set. Moreover, if M computes a function f, then f is also computable. However, many computable sets and functions cannot be computed by a finite automaton.

Example 2.5. The function $f(x) = x^2$ is not computable by a finite state automaton. Suppose that M computes x^2 on input x, where we have appended a sufficient number of 0's to the end of the input so that we can output x^2 (recall that each input bit yields at most one output bit). In particular, for any n, M will output $0^{2n}1$ on input 0^n1, since $f(2^n) = (2^n)^2 = 2^{2n}$. Thus, on input 0^n1, after reading the first $n+1$ bits, M needs to examine at least n additional 0's and write at least n additional 0's before it finishes the output with a final 1. Now suppose that M has k states and let $n > k+1$. Let s_j be the state of the machine after reading 0^n10^j. Then there must be some $i, j \leq k+1 < n$ such that $s_i = s_j$. Furthermore, every time M transitions from state s_i upon reading a 0, M must output a 0. But the machine is essentially stuck in a *loop* and hence can only print another 0 after reading 0^n10^n when it needs to print the final 1.

3. Turing Machines

A Turing machine is a simple model of a computer that is capable of computing *any* function that can be computed. It consists of these items:

(1) a finite state control component with a finite number of read/write heads; and
(2) a finite number of unbounded memory tapes (one or more for the input(s), one for the output, and the rest for scratchwork), each of which is divided into infinitely many consecutive squares in which symbols can be written.

Furthermore, it must satisfy these conditions:

(1) there is a specified initial state q_0 and a specified final q_H, or *halting*, state; and
(2) each read/write head reads one cell of each tape and either moves one cell to the left (L), moves one cell to the right (R), or stays stationary (S) at each stage of a computation.

The notion of Turing machine is formalized in the following definition.

Definition 3.1. *A k-tape* Turing machine *M for an alphabet Σ consists of*

(i) *a finite set $Q = \{q_0, \dots, q_n\}$ of states;*
(ii) *an alphabet Σ;*
(iii) *a transition function $\delta : Q \times \Sigma^k \to \Sigma^k \times \{L,R,S\}^k$;*
(iv) *$q_0 \in Q$ is the start state; and*
(v) *$q_H \in Q$ is the halting or final state.*

A *move* of M in a given state q_i, scanning the symbols $a_1, \dots, a_k$ on the k tapes, where $\delta(q_i, a_1, \dots, a_k) = (q_j, b_1, \dots, b_k, D_1, \dots, D_k)$, consists of the following actions:

(1) switching from state q_i to state q_j;
(2) writing b_i (and thus erasing a_i) on tape i; and
(3) moving the head on tape i in the direction D_i.

A *computation* always begins with

(i) the machine in state q_0;
(ii) some finite input on each of the input tapes; and
(iii) each of the input heads scanning the first symbol on each of the input tapes.

The *configuration* of a machine at a given stage of a computation consists of

(i) the current state of the machine;

(ii) the contents of each tape; and
(iii) the location of each of the heads on each tape.

M machine *halts* after n moves if it transitions to the halting state in the n-th stage of the computation. A machine M *accepts* a word w, denoted $M(w)\downarrow$, if the machine halts (i.e. ends up in the halting state) when given the input w. In this case, we say M *halts* on the input w. Otherwise, we say that M *diverges* on input w, denoted $M(w)\uparrow$.

It is an essential feature of Turing machines that they may fail to halt on some inputs. This gives rise to the *partial computable function* f_M which has domain $\{w : M(w)\downarrow\}$. That is, $f_M(w) = y$ if and only if M halts on input w and y appears on the output tape when M halts, meaning that the output tape contains y surrounded by blanks on both sides. (For the sake of elegance, we may insist that the first symbol of y is scanned at the moment of halting.) Sometimes we will write the value $f_M(w)$ as $M(w)$.

Example 3.2. We define a Turing machine M_1 that computes $x + y$. There are two input tapes and one output tape. The numbers x and y are written on separate input tapes in reverse binary form. M has the states: the initial state q_0, a *carry* state q_1 and the halting state q_H. The two input tapes are read simultaneously in the form a/b. We need to consider blanks squares as symbols # in case one input is longer and/or there is a carry at the end. The behavior of M_1 is summed up in the following table.

State	Read	Write	Move	New State
q_0	0/0	0	R	q_0
q_0	0/#	0	R	q_0
q_0	#/0	0	R	q_0
q_0	0/1	1	R	q_0
q_0	1/0	1	R	q_0
q_0	1/#	1	R	q_0
q_0	#/1	1	R	q_0
q_0	1/1	0	R	q_1
q_0	#/#	#	S	q_H
q_1	0/0	1	R	q_0
q_1	0/#	1	R	q_0
q_1	#/0	1	R	q_0
q_1	0/1	0	R	q_1
q_1	1/0	0	R	q_1
q_1	1/#	0	R	q_1
q_1	#/1	0	R	q_1
q_1	1/1	1	R	q_1
q_1	#/#	1	S	q_H

Example 3.3. We roughly describe a Turing machine M_2 that computes $x \cdot y$. Again there are two input tapes and one output tape. The idea is that if $y = \sum_{i \in I} 2^i$ for some finite $I \subseteq \mathbb{N}$, then $x \cdot y = \sum_{i \in I} 2^i \cdot x$. For each i, $2^i \cdot x$ has the form of i 0's followed by x. For example, if $x = 1011$ (thirteen), then $4 \cdot x = 001011$ (fifty-two).

To multiply 1011 by 100101 we add

$$2^0 \cdot 13 + 2^3 \cdot 13 + 2^5 \cdot 13 = 1011 + 0001011 + 000001011.$$

We begin in state q_0 with x and y each written on one of the input tapes (tapes 1 and 2) in reverse binary notation, and all three reader heads lined up. We first add a terminal 0 to the end of y, which is necessary to ensure that our computation will halt.

Suppose there are k initial 0's in y. For each such 0, we replace it with a #, write a 0 on tape 3 in the corresponding cell, move the heads above tapes 2 and 3 one cell to the right, and stay in state q_0.

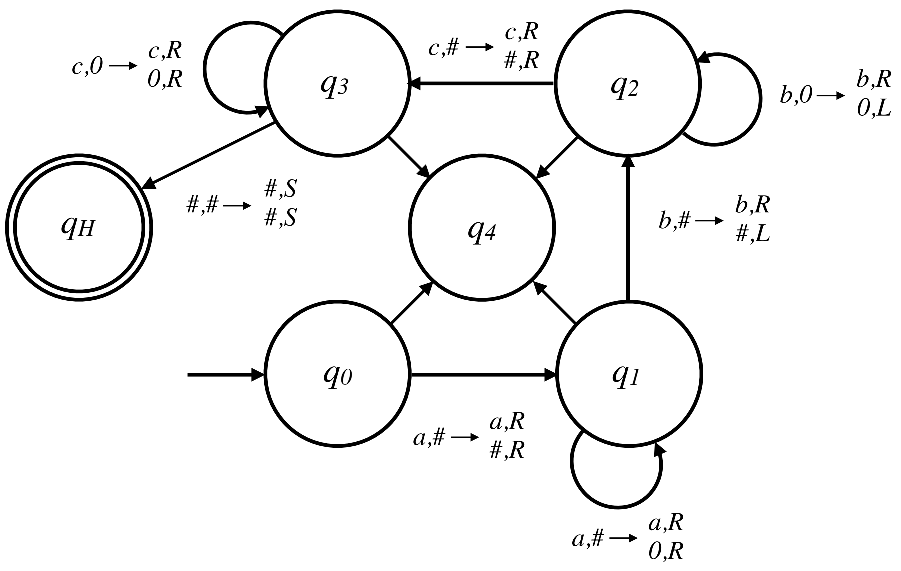
When we encounter the first 1 of y, we replace this 1 with a # and transition to a state q_C in which we copy x to tape 3 (the output tape), beginning in the i-th cell of this tape. As the contents of tape 1 are being copied onto tape 3, we also require that head above tape 2 moves in the same directions as the head above tapes one and three until we encounter the first # on tape one. (One can verify that the heads above tapes 2 and 3 will always be lined up). We then transition to a state in which we reset the position of the tape 1 head to scanning the first bit of x, and then the tape 2 head is scanning the leftmost bit in y that has not been erased (i.e. replaced with a #). As above, we require that the tape 3 head moves in the same directions as the tape 2 head during this phase (which will ensure that we begin copying x in the correct place on the tape if the next bit of y happens to be a 1. We then transition back to state q_0 and continue as before.

Suppose that the next bit of y that is equal to 1 is the i-th bit of y. Then we start the addition by adding the first bit of x to the i-th bit of the output tape. We note that we may need to utilize a carry state during the addition. When we reach the # at the end of y, the computation is complete, and we transition to the halting state. For an example of the product $7 \ldots 8 = 7 \cdot 8 = 56$, see the supplementary document TuringMachineMultiplication.pdf.

Like finite state automata, Turing machines can be represented a state diagram.

Example 3.4. There is a Turing machine that accepts every string in the set $\{a^n b^n c^n \mid n > 0\}$ over the alphabet $\Sigma = \{a, b, c, 0, \#\}$.

Here we will use the "0" as a special marking symbol, although we can do without it (and use "a" instead). The following state diagram gives a Turing machine which accepts this every string in the above set.



Observe that q_4 functions as a reject state. Any input not listed on any of the edges will cause a transition to state q_4. To see that this Turing machine accepts the desired set, see the supplementary document Turingmachine-a4b4c4.pdf for an example with input $aaaabbbbcccc$.

Example 3.5. We define a Turing machine M that computes the function $f(x) = |x|$ (the length of a string x). There are two tapes, the input tape and the output tape. The input tape is read-only but we allow writing on the output tape. Let the input alphabet be $\Sigma = \{a\}$. Let α/β indicate that the machine is currently reading α on the input tape and β on the output tape. Similarly D_1/D_2 indicates that the head on the input tape moves in direction D_1 while the head on the output tape moves in direction D_2. The idea is to add one to the output tape after reading each symbol of the input tape. State q_1 arises when we need to add one to the output by carrying. State q_2 simply brings the output tape back to the first bit. Certain transitions are omitted from the table since they lead to divergent computation (we will assume that incorrect inputs will immediately cause a transition to a reject state). For example, we only get to state q_1 when we have read a on the first tape and we continute to read that a, so that the input $\#/0$ is not a legal input when we have reached state q_1.

The following table describes the behavior of the three main states.

state	read	write	move	new state
q_0	$a/\#$	$a/1$	R/S	q_0
q_0	$a/0$	$a/1$	R/S	q_0
q_0	$a/1$	$a/0$	S/R	q_1
q_0	$\#/0$	$\#/0$	S/S	q_H
q_0	$\#/1$	$\#/0$	S/S	q_H
q_0	$\#/\#$	$\#/0$	S/S	q_H
q_1	$a/\#$	$a/1$	S/L	q_2
q_1	$a/0$	$a/1$	S/L	q_2
q_1	$a/1$	$a/0$	S/R	q_1
q_2	$a/\#$	$a/\#$	R/R	q_0
q_2	$a/0$	$a/0$	S/L	q_2
q_2	$a/1$	$a/1$	S/L	q_2

We now prove some general facts about certain kinds of languages that are central to the study of Turing machines.

For a fixed alphabet Σ, a *language* is simply a set $L \subseteq \Sigma^*$ (recall that Σ^* is the collection of all finite sequences of elements of Σ).

Definition 3.6. *A language L is said to be* Turing semicomputable *if there is a Turing machine M such that $L = \{w : M(w)\downarrow\}$. L is a* Turing computable *language if the characteristic function of L is Turing computable.*

Example 3.7. Here is a simple Turing machine M such that $M(w)\downarrow$ if and only if w contains a 0. In state q_0, M moves right and remains in state q_0 upon reading a 1 or a blank. M immediately halts upon reading a 0.

Proposition 3.8. *Every Turing computable language is also Turing semicomputable.*

Proof. Let M be a Turing machine that computes the characteristic function of L. We modify M to define a machine M' as follows. First we introduce new states q_A and q_B. Replace any transition that goes to the halting state q_H with a transition that goes the the state q_A. For the q_A state, add two transitions. If the output tape reads 1, then transition to the halting state q_H. If the output tape reads 0, then move the output tape head one cell to the right and transition to state q_B. In state q_B, move the output tape head one cell to the left and return to state q_A. Then $M'(w)$ will halt if and only if $M(w) = 1$ and will endlessly alternate between states q_A and q_B if and only if $M(w) = 0$. □

Proposition 3.9. *L is Turing computable if and only if both L and its complement are Turing semicomputable.*

Proof. First observe that if $L \subseteq \Sigma^*$ is Turing computable, then $\Sigma^* - L$ is Turing computable. Indeed, if M computes the characteristic function of L, then define M' to be the machine that behaves exactly like M except that for $i = 0, 1$ whenever M writes i on its output tape, M' writes $1 - i$ on its output tape. It follows from Proposition 3.8 that if L is Turing computable, then both L and its complement are semicomputable.

Now suppose that $L = \{w : M_0(w)\downarrow\}$ and that $\Sigma^* - L = \{w : M_1(w)\downarrow\}$ for two Turing machines M_0 and M_1. We define a Turing machine M such that the function f_M computed by M is the characteristic function of L. Suppose for the sake of simplicity that M_0 and M_1 each have one input tape and have no output tape. Then M will have one input tape, two scratch tapes, and one output tape. The states of M will include pairs (q, q') where q is a state of M_0 and q' is a state of M_1. Given w on the input tape of M, M will begin by copying w onto each of the scratch tapes and transitioning to the pair (q_0, q_0') of initial states of M_0 and M_1. On the first scratch tape, M will simulate M_0, while on the second scratch tape, M will simulate M_1. Eventually M will enter a state of one of the two forms: (q_H, r), where q_H is the halting state of M_0 and r is a non-halting state of M_1, or (q, q_H'), where q_H' is the halting state of M_1 and q is a non-halting state of M_0.

— If M enters a state of the form (q_H, r), this means that the machine M_0 has halted on w, and hence $w \in L$. M will thus write a 1 on its output tape and transition to its halting state.
— If M enters a state of the form (q, q_H'), this means that the machine M_1 has halted on w, and hence $w \notin L$. M will thus write a 0 on its output tape and transition to its halting state.

Note that M will never enter the state (q_H, q_H'), since M_0 and M_1 can never accept the same word. It thus follows that M computes the characteristic function of L. □

Given two Turing machines M_0 and M_1, we write $M_0(x) \simeq M_1(x)$ to mean that (i) $M_0(x)\downarrow$ if and only if $M_1(x)\downarrow$ and (ii) $M_0(x)\downarrow$ implies that $M_0(x) = M_1(x)$.

Theorem 3.10. *Fix a finite alphabet $\Sigma = \{0, 1, q, \#, *, L, R, S, H, \downarrow\}$. There is a* universal *Turing machine $U : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that, for any Turing machine $M : \{0,1\}^* \to \{0,1\}^*$, there exists $w_M \in \Sigma^*$ such that, for all inputs $x \in \{0,1\}^*$, $U(w_M, x) \simeq M(x)$.*

We will only sketch the main ideas in the proof of Theorem 3.10. To simplify matters, we will use the following lemma.

Lemma 3.11. *Let M be a k-tape Turing machine for some $k > 1$. Then there is a single-tape Turing machine M' such that $M(x) \simeq M'(x)$ for all $x \in \Sigma^*$.*

Proof of Theorem 3.10. (Sketch) We define a universal Turing machine with two input tapes, a scratch tape, and an output tape. For each machine M, we would like to define a word $w_M \in \Sigma^*$ that encodes all of the information of M. We will let w_M be the entire transition table of M written as one long string in the alphabet Σ^*, with a $*$ separating each entry on a given row of a table and $**$ separating each row of the table. The string $q0$ will stand for the initial state, qH will stand for the halting state, and all other states will be coded by a q followed by a finite string of 1s ($q1$, $q11$, $q111$, etc.)

Now, given the code w_M for a machine M, to compute $U(w_M, x)$ (i.e. $M(x)$), U proceeds by writing the initial configuration of M with input x on its scratch tape. Suppose, for example that $x = 000$. Then U will write

$$q0{*}\downarrow 000$$

where the $\downarrow$ specifies that the reader head is above the first 0 on the input tape of M. To proceed, U simply consults the transition table w_M written on its first input tape and writes the resulting configuration of M after one move on its scratch tape. Continuing the example from above, if the first move of M is to change to state q_1, replace the first 0 of x with a 1 and move the head one cell to the right, the resulting configuration will be

$$q1 * 1 \downarrow 00$$

Continuing in this way, if $M(x)\downarrow$, then U will eventually come to a stage in which a halting configuration is written on its scratch tape. In this halting configuration, the value $y = M(x)$ will be written, and so U can simply copy this value y to its output tape and transition to its own halting state. Lastly, if $M(x)\uparrow$, then $U(w_M, x)\uparrow$.

□

It is important to note that our use of the alphabet Σ in the definition of a universal Turing machine is not strictly necessary. For instance, we can also define a universal Turing machine $U : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ by representing each of the symbols in Σ as a unique binary string. Some caution is necessary to make sure the coding is unambiguous. We will not discuss the details of such a coding here, but we will assume that for each Turing machine M, there is some unique $w_M \in \{0,1\}^*$ that codes M. Moreover, we will assume that every $x \in \{0,1\}^*$ codes some Turing machine, which we will write as M_x. (We will justify these assertions in Chapter 7 when we discuss Gödel numbering.)

Programs nearly always have bugs, so they may not do want we want them to do. The problem of determining whether a given Turing machine M halts on input string w is the *Halting Problem*. Let us define the halting set to be $H = \{(x, y) \in \{0,1\} * \times\{0,1\}^* : M_x(y)\downarrow\}$. Observe that H is semicomputable: $(x, y) \in H$ if and only if $U(x, y)\downarrow$. By contrast, we have the following.

Theorem 3.12. *The Halting Problem is not computable.*

Proof. We will show that the complement of H is not semicomputable, so that by Proposition 3.9, H is not computable. Suppose by way of contradiction that there is a Turing machine M such that, for all $x, y \in \{0,1\}^*$,

$$M(x, y)\downarrow \iff M_x(y)\uparrow.$$

We can define a Turing machine N so that $N(w) \simeq M(w, w)$. Then

$$N(w)\downarrow \iff M_w(w)\uparrow.$$

But this Turing machine N must have some code e. So for all w,

$$M_e(w)\downarrow \iff M_w(w)\uparrow.$$

The contradiction arises when $w = e$. □

Thus there exists a set which is semicomputable but not computable.

4. Recursive Functions

In this section, we define the primitive recursive and the (partial) recursive functions and show that they are all Turing computable. Each function f maps from $\mathbb{N}^k$ to $\mathbb{N}$ for some fixed k (the *arity* of f).

Definition 4.1. *The collection of* primitive recursive functions *is the smallest collection $\mathscr{F}$ of functions from $\mathbb{N}^k$ to $\mathbb{N}$ for each $k > 0$ that includes the following* initial functions

(1) *the constant function $c(x) = 0$,*
(2) *the successor function $s(x) = x + 1$,*
(3) *the projection functions $p_i^k(x_1, \ldots, x_k) = x_k$ for each $k \in \mathbb{N}$ and $i = 1, \ldots, k$,*

and are closed under the following schemes for defining new functions:

(4) *(composition) if $f : \mathbb{N}^k \to \mathbb{N}$ and $g_i : \mathbb{N}^j \to \mathbb{N}$ for $i = 1, \ldots, k$ (where $j, k > 0$) are in $\mathscr{F}$, then the function $h : \mathbb{N}^j \to \mathbb{N}$ defined by*

$$h(x_1, \ldots, x_j) = f(g_1(x_1, \ldots, x_j), \ldots, g_k(x_1, \ldots, x_j))$$

is in $\mathscr{F}$, and

(5) *(primitive recursion) if* $g : \mathbb{N}^k \to \mathbb{N}$ *and* $h : \mathbb{N}^{k+2} \to \mathbb{N}$ *are in* $\mathscr{F}$, *then the function* $f : \mathbb{N}^{k+1} \to \mathbb{N}$ *defined by*

$$f(0, x_1, \dots, x_k) = g(x_1, \dots, x_k)$$
$$f(n+1, x_1, \dots, x_k) = h(n, x_1, \dots, x_k, f(n, x_1, \dots, x_k))$$

is in $\mathscr{F}$.

Example 4.2.

(1) For any constant $c \in \mathbb{N}$, the function $h(x) = c$ is primitive recursive. The proof is by induction on c. For $c = 0$, this follows from the fact that the initial function $c(x) = 0$ is primitive recursive. Supposing that $g(x) = c$ is primitive recursive and using the fact that $s(x) = x + 1$ is primitive recursive, we can use composition to conclude that $h(x) = s(g(x)) = g(x) + 1 = c + 1$ is primitive recursive.

(2) For any k and any c, the constant function $f : \mathbb{N}^k \to \mathbb{N}$ with $f(x_1, \dots, x_k) = c$ is primitive recursive. We have $h(x) = c$ by (1) and we have $p^k_1(x_1, \dots, x_k) = x_1$ as a basic function, so that $f(x_1, \dots, x_k) = h(p^k_1(x_1, \dots, x_k)) = h(x_1) = c$ is also primitive recursive.

(3) The addition function $f(x, y) = x + y$ is primitive recursive. Let $g(y) = p^1_1(y)$ and $h(x, y, z) = c(p^3_3(x, y, z))$. Then f is given by

$$f(0, y) = g(y) = y$$
$$f(n+1, y) = h(n, y, f(n, y)) = f(n, y) + 1.$$

(4) The *predecessor* function $f(x) = x \dot{-} 1 = \begin{cases} x - 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \end{cases}$ is primitive recursive. Let $g(x) = c(x)$ and $h(x, y) = p^2_1(x, y)$. Then f is given by

$$f(0) = g(y) = 0$$
$$f(n+1) = h(n, f(n)) = n.$$

(5) The truncated subtraction function $f(x, y) = \begin{cases} y \dot{-} x, & \text{if } x \le y \\ 0, & \text{otherwise} \end{cases}$ is primitive recursive. Let $g(y) = p^1_1(y)$ and $h(x, y, z) = z \dot{-} 1$. Then f is given by

$$f(0, y) = g(y) = y$$
$$f(n+1, y) = h(n, y, f(n, y)) = f(n, y) \dot{-} 1.$$

(6) The function $sg = \begin{cases} 0, & \text{if } x = 0 \\ 1, & \text{if } x > 0 \end{cases}$ is clearly primitive recursive.

(7) The multiplication function $f(x, y) = x \cdot y$ is primitive recursive. Let $g(x) = c(x) = 0$ and let $h(x, y, z) = p^3_3(x, y, z) + p^3_2(x, y, z)$. Then f is given by

$$f(0, y) = g(y) = 0$$
$$f(n+1, y) = h(n, y, f(n, y)) = f(n, y) + y.$$

We can extend the collection of primitive recursive functions to the collection of partial recursive functions by adding one additional scheme for defining new functions from previously defined ones. It is this scheme that allows for the possibility that a function be undefined on a given input. Given a function $f : \mathbb{N}^k \to \mathbb{N}$, if f is defined on input $(x_1, \dots, x_k)$, we write $f(x_1, \dots, x_k)\downarrow$; if f is undefined on $(x_1, \dots, x_k)$, we write $f(x_1, \dots, x_k)\uparrow$.

Definition 4.3. *The collection of* partial recursive functions *is the smallest collection $\mathscr{F}$ of functions from $\mathbb{N}^k$ to $\mathbb{N}$ for each $k > 0$ that includes the primitive recursive functions and is closed under the following scheme:*

(6) *(unbounded search) if $g : \mathbb{N}^{k+1} \to \mathbb{N}$ is in $\mathscr{F}$, then the function $f : \mathbb{N}^k \to \mathbb{N}$ defined by*
$f(x_1, \dots, x_k) =$ the least n such that $g(n, x_1, \dots, x_k) = 0$ and $g(i, x_1, \dots, x_k)\downarrow$ for $i = 0, \dots n$ (and $f(x_1, \dots, x_k)\uparrow$ otherwise)
is in $\mathscr{F}$.

We will refer to total recursive functions simply as recursive functions.

We say that f is defined from the total function g by *bounded search* if $f(n, x)$ equals the least $i < n$ such that $g(i, x) = 0$ and otherwise $f(n, x) = n$. Certainly if g is a recursive function, then f will be recursive function. We note that if we add a scheme of *bounded search* to the collection of primitive recursive functions, we do not add any new functions.

Lemma 4.4. *If g is primitive recursive and f is defined from g by bounded search, then f is primitive recursive.*

Proof. We have $f(0, x) = 0$ and for each n,

$$f(n+1, x) = \begin{cases} f(n, x) & \text{if } g(f(n, x), x) = 0 \\ n+1 & \text{otherwise} \end{cases}.$$

□

The collection of partial recursive functions is equivalent to the collection of Turing computable functions, in the sense that every partial recursive function can be computed by a Turing machine, and every Turing computable function is partial recursive. We will prove one direction of this equivalence.

Theorem 4.5. *Every partial recursive function can be computed by a Turing machine.*

Proof. The proof is by induction on the family of partial recursive functions. For the base case, it is clear that the initial functions are Turing computable. We now verify that the schemes of composition, primitive recursion, and unbounded search yield Turing computable functions when applied to Turing computable functions.

Composition. For simplicity let $f(x) = h(g(x))$ where h is computable by machine M_h and g is computable by machine M_g. Assume without loss of generality that each machine uses one input tape and one output tape and that the sets of states of the two machines are disjoint. We define a machine M_f that computes f with six tapes:

(1) the input tape;
(2) a scratch tape to serve as the input tape of g;
(3) a scratch tape to serve as the output tape of g;
(4) a scratch tape to serve as the input tape for h;
(5) a scratch tape to serve as the output tape of h; and
(6) an output tape.

M will have the states of g and h together plus a few new states to handle the transfer from M_g to M_h. The transition function for M_g will be changed so that instead of halting when the output $g(x)$ is ready, M will go into a subroutine which copies from the g-output tape to the h-input tape and then hands over the controls to M_h. The halting states of M will be the halting states of M_h.

Primitive Recursion. For simplicity let $f(0,x) = g(x)$ and $f(n+1,x) = h(n,x,f(n,x))$. Let M_g compute g and let M_h compute h. Assume without loss of generality that each machine uses one input tape and one output tape and that the sets of states of the two machines are disjoint. We define a machine M that computes $f(n,x)$ with nine tapes:

(1) the input tape for n;
(2) the input tape for x;
(3) a scratch tape to serve as the input tape for g;
(4) a scratch tape to serve as the output tape of g;
(5) a tape to keep track of the ongoing value of $m < n$;
(6) a tape to keep track of the ongoing value of $f(m,x)$;
(7) a scratch tape to serve as the input tape for h;
(8) a scratch tape to serve as the output tape of $h(m,x,f(m,x))$; and
(9) an output tape.

M will have the states of g and h together plus a few new states to handle the transfer from M_g to M_h and the ongoing recursion. The transition function for M_g will be changed so that instead of halting when the output $g(x)$ is ready, M will copy the value $g(x)$ onto tape (6), write $m = 0$ onto tape (5), and then hand over control to M_h. The inputs for M_h are found on tapes (2), (5) and (6). M_h uses these to compute $h(m,x,f(m,x)) = f(m+1,x)$. When M_h is ready to halt and give its output, M does the following:

(i) M compares m from tape (5) with n from tape (1); if $n = m+1$, then the value on tape (8) equals the desired $f(n,x)$, so M copies this to tape (9) and halts.
(ii) Otherwise, M erases tape (6) and then copies the value from tape (8) onto tape (6).
(iii) Then M adds one to the value of m on tape (5), erases tapes (7) and (8) and hands control back to M_h again.

Unbounded Search. For simplicity let $f(x)$ = the least n such that $g(n,x)\!\downarrow = 0$ and for all $i \leq n$ $g(i,x)\!\downarrow$. Let M_g compute g using one input tape and one output tape.

We define a machine M that computes $f(x)$ with five tapes:

(1) the input tape for x;
(2) a tape for the ongoing value of n;
(3) a scratch tape to serve as the input tape for g;
(4) a tape to keep track of the ongoing value of $g(n,x)$; and
(5) an output tape.

M will have the states of g plus a few new states to handle the the ongoing computations of $g(n,x)$. M begins by writing $n = 0$ on tape (2) and handing control to M_g. The transition function for M_g will be changed so that instead of halting when the output $g(x)$ is ready, M will do the following:

(i) Compare the value $g(n,x)$ from tape (4) with 0. If $g(n,x) = 0$, then the value n on tape (2) equals the desired $f(x)$, so M copies this to tape (5) and halts.
(ii) Otherwise, M increments the value of n on tape (2), erases tapes (3) and (4) and hands control back to M_g again.

□

For the other direction, we need to associate to each Turing machine a natural number. This is not difficult to do, as we've already associated to each Turing machine some $x \in \{0,1\}^*$. Furthermore, for each machine M, we can represent each configuration of M as a natural number, and in fact we can represent each computation (given as a sequence of configurations) as a natural number. We will not discuss the details of such representations. As long as we code machines, configurations, and computations in a reasonable way, we have the following result.

Theorem 4.6 (Kleene's Normal Form Theorem). *There is a predicate $T(e,x,y) \subseteq \mathbb{N}^3$ with a primitive recursive characteristic function and a primitive recursive function $U : \mathbb{N} \to \mathbb{N}$ such that for each Turing computable function $\phi : \mathbb{N} \to \mathbb{N}$, there is some e such that if y is the least such that $T(e,x,y)$ holds, then*

$$\phi(x) = U(y).$$

Here $T(e,x,y)$ asserts that y is the code of some Turing computation of the e-th Turing machine with input x. If $T(e,x,y)$ holds for some y, then to compute $U(y)$ we simply run this Turing computation and convert the value of the output tape into a natural number.

All of the formalizations of the intuitive notion of a computable number-theoretic function has given rise to the same collection of functions. For this and other reasons, the community of mathematical logicians have come to accept the *Church-Turing thesis*, which is the claim that the collection of Turing computable functions is the same as the collection of intuitively computable functions. In practice, the Church-Turing thesis has two main consequences:

(1) if we want to show that a given problem cannot be solved by any algorithmic procedure, it suffices to show that solutions to the problem cannot be computed by any Turing computable functions;
(2) to show that a given function is computable, it suffices to give an informal description of an effective procedure for computing the values of the function.

Given the equivalence of the various notions of computable function, we will hereafter refer to the formal notions as *computable functions* and *partial computable functions* (as opposed to *recursive functions* and *partial recursive functions*).

We now recast the earlier definitions of computability and semi-computability in terms of natural numbers and introduce a new notion, namely, computable enumerability.

Definition 4.7. *Let $A \subseteq \mathbb{N}$.*

(1) *A is said to be* primitive recursive *if the characteristic function χ_A is primitive recursive.*
(2) *A is said to be* computable *if the characteristic function χ_A is computable.*
(3) *A is said to be* semi-computable *if there is a partial computable function ϕ such that $A = dom(\Phi)$.*
(4) *A is said to be* computably enumerable *if there is some computable function f such that $A = ran(f)$. That is, A can be enumerated as $f(0), f(1), f(2), \dots$.*

Example 4.8. The set of even numbers is primitive recursive since its characteristic function may be defined by $f(0) = 1$ and $f(n+1) = 1 - f(n)$.

Example 4.9. Define the functions Q and R as follows. Let $Q(a,b)$ be the quotient when b is divided by $a+1$ and let $R(a,b)$ be the remainder, so that $b = Q(a,b)\cdot(a+1) + R(a,b)$. Then both Q and R are primitive recursive. That is, $Q(a,b)$ is the least $i \leq b$ such that $i \cdot (a+2) \geq b$ and $R(a,b) = b - Q(a,b)\cdot(a+1)$.

Example 4.10. The relation $x \mid y$ (x divides y) is primitive recursive, since $x \mid y$ if and only if $(\exists q < y+1)\, x \cdot q = y$.

Example 4.11. The set of prime numbers is primitive recursive and the function P which enumerates the prime numbers in increasing order is also primitive recursive. To see this, note that $p > 1$ is prime if and only if $(\forall x < p)[x \mid p \to p = 1]$. Now we know that for any prime p, there is another prime $q > p$ with $q < p! + 1$. By one of the exercises, the factorial function is primitive recursive. Then we can recursively define P by $P(0) = 2$ and, for all i,

$$P(i+1) = (\text{least } x < P(i+1)! + 1)\ x \text{ is prime.}$$

We conclude this chapter with the following result.

Theorem 4.12. *A is computably enumerable if and only if A is semicomputable.*

Proof. Suppose first that A is computably enumerable. If $A = \emptyset$, then certainly A is semicomputable, so we may assume that $A = rng(f)$ for some computable function f. Now define the partial computable function ϕ by $\phi(x) = (\text{least } n) f(n) = x$ for $x \in \mathbb{N}$. Then $A = dom(\phi)$, so that A is semicomputable.

Next suppose that A is semicomputable and let ϕ be a partial computable function so that $A = dom(\phi)$. If A is empty, then it is computably enumerable. If not, select $a \in A$ and define the computable function f by

$$f(2^s \cdot (2m+1)) = \begin{cases} m, & \text{if } \phi(m)\downarrow \text{ in } < s \text{ steps,} \\ a, & \text{otherwise.} \end{cases}$$

Then $A = rng(f)$, so that A is computably enumerable. □

FOUNDATIONS OF MATHEMATICS
CHAPTER 6: DECIDABLE AND UNDECIDABLE THEORIES

1. Introduction

In this chapter we bring together the two major strands that we have considered thus far: logical systems (and, in particular, their syntax and semantics) and computability theory. We will briefly discuss decidable and undecidable logical systems, but our primary focus will be decidable and undecidable first-order theories.

1.1. **Gödel numbering.**

In order to apply the tools of computability theory to the study of various logical systems and first-order theories, we need to represent the objects such as formulas and proofs as objects that we can give as input to computable functions (such as strings over some fixed alphabet or natural numbers). Here we will code formulas as natural numbers, which can then be represented as binary strings. To do so, we first need to code each symbol in our logical language as a natural number. Below are two such coding schemes for the symbols used in propositional and predicate logic.

A coding of the symbols of proposition logic:

symbol	code	symbol	code
(	1	$\vee$	5
)	2	$\rightarrow$	6
&	3	$\leftrightarrow$	7
$\neg$	4	P_i	$8+i$

A coding of the symbols of predicate logic:

symbol	code	symbol	code
(	1	$=$	8
)	2	$\exists$	9
&	3	$\forall$	10
$\neg$	4	v_i	$11+4i$
$\vee$	5	c_i	$12+4i$
$\rightarrow$	6	P_i	$13+4i$
$\leftrightarrow$	7	F_i	$14+4i$

Given a formula ϕ (of propositional logic or of predicate logic), suppose that the codes of the symbols that make up ϕ are $n_1, \dots, n_k$ (so that, for instance, the codes of the symbols in the propositional formula P_0 & P_1 are 8, 3, 9). Then we will represent ϕ by the number

$$p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$$

where p_i is the i-th prime number for $i = 1, \dots, k$. We will refer to this number as the *Gödel number* of ϕ.

2. Decidable vs. Undecidable Logical Systems

Let us consider some examples of decidable and undecidable logical systems.

Definition 2.1. *A logical system is decidable if there is an effective procedure that, given the Gödel number of a sentence ϕ, outputs 1 if ϕ is logically valid and outputs 0 if ϕ is not logically valid.*

For a given logical system such as propositional logic or predicate logic, we say that the *decision problem* for this system is the problem of determining whether a given formula is logically valid. Moreover, for a decidable logical system, we say that its decision problem is *solvable*; similarly, for an undecidable logical system, we say that its decision problem is *unsolvable*.

We now turn to our examples.

Example 2.2. *The decision problem for propositional logic is solvable.*

The method of truth-tables provides an algorithm for determining whether a given propositional formula ϕ is logically valid.

Example 2.3. *The decision problem for monadic predicate logic is solvable.*

Monadic predicate logic is first-order logic with only 1-place predicates such as $R(x), B(y)$, etc. A key result needed to show the decidability of monadic predicate logic is the following:

Theorem 2.4. *If ϕ is a sentence of monadic predicate logic consisting of k distinct monadic predicates and r distinct variables, then if ϕ is satisfiable, it is satisfiable in a model of size at most $2^k \cdot r$.*

As a corollary, we have:

Corollary 2.5. *If ϕ is a sentence of monadic predicate logic consisting of k distinct monadic predicates and r distinct variables and is not logically valid, then there is a model of size at most $2^k \cdot r$ in which $\neg\phi$ is satisfied.*

Thus to determine if a sentence of monadic predicate logic is logically valid, we must check to see whether ϕ is true in all models of cardinality less than some finite bound, which can be done mechanically.

We now turn to some examples of undecidable logical systems.

Example 2.6. *The decision problem for predicate logic is unsolvable.*

To show this, one can prove that, given the transition table of a Turing machine M, there is a finite set Γ_M of $\mathscr{L}$-sentences in some first-order language $\mathscr{L}$ and an additional collection of $\mathscr{L}$-sentences $\{\phi_n\}_{n\in\mathbb{N}}$ such that for every n,

$$\Gamma_M \models \phi_n \Leftrightarrow M(n)\downarrow.$$

By the Completeness Theorem, it follows that

$$\Gamma_M \vdash \phi_n \Leftrightarrow M(n)\downarrow,$$

which is equivalent to

$$\vdash \Gamma_M \rightarrow \phi_n \Leftrightarrow M(n)\downarrow.$$

Now suppose there is an effective procedure that, given the Gödel number of any first-order sentence ϕ, will output a 1 if $\vdash \phi$ and outputs a 0 otherwise. Then for all sentences of the form $\Gamma_M \to \phi_n$, this procedure will output a 1 if and only if $\vdash \Gamma_M \to \phi_n$, which holds if and only if $M(n)\downarrow$. But this contradicts the unsolvability of the halting problem, and thus there can be no such effective procedure.

Example 2.7. *The decision problem for dyadic first-order logic is unsolvable.*

Dyadic predicate logic is first-order logic with only 2-place predicates such as $R(x,y), B(y,z)$, etc. One can in fact show that if our language includes just *one* 2-place predicate, this is sufficient to create a collection of sentences for which the decision problem is unsolvable.

3. Decidable Theories

In this section we identify a sufficient condition for a theory T to have an algorithm that enables us to determine the consequences of T. Then we will provide a specific example of a theory satisfying this condition, namely, the theory of dense linear orders without endpoints.

Hereafter, let us fix a first-order language $\mathscr{L}$. Given a set of $\mathscr{L}$-sentences S, we will often identify it with the set of Gödel numbers of the sentences in S. Thus, when we say that a set S of formulas is, say, computably enumerable, we really mean that the set of Gödel numbers of the formulas in S is computably enumerable.

Definition 3.1. *An $\mathscr{L}$-theory Γ is decidable if there is an algorithm to determine for any $\mathscr{L}$-sentence φ whether $\Gamma \vdash \varphi$ or $\Gamma \nvdash \varphi$.*

Recall that for an $\mathscr{L}$-theory Γ, $Th(\Gamma) = \{\phi : \Gamma \vdash \phi\}$.

Definition 3.2. *Let Γ be an $\mathscr{L}$-theory.*

(i) *Γ is finitely axiomatizable if there is a finite set Σ of $\mathscr{L}$-sentences such that $Th(\Sigma) = Th(\Gamma)$.*
(ii) *Γ is computably axiomatizable if there is a computable set Σ of $\mathscr{L}$-sentences such that $Th(\Sigma) = Th(\Gamma)$.*

Lemma 3.3. *If Γ is a computably axiomatizable $\mathscr{L}$-theory, then $Th(\Gamma)$ is computably enumerable.*

Proof Sketch. Let Σ be a computable collection of $\mathscr{L}$-sentences such that $Th(\Sigma) = Th(\Gamma)$. Since the collection of all possible proofs with premises from Σ is computably enumerable, the collection of $\mathscr{L}$-sentences that are the conclusion of some proof with premises from Σ is also computably enumerable. This collection of $\mathscr{L}$-sentences is precisely the collection of consequences of Σ and hence of Γ. It follows that $Th(\Gamma)$ is computably enumerable. □

Theorem 3.4. *If Γ is a computably axiomatizable, complete $\mathscr{L}$-theory, then Γ is decidable.*

Sketch of Proof. By the previous lemma, $Th(\Gamma)$ is computably enumerable. Let f be a total computable function whose range is $Th(\Gamma)$. Since Γ is complete, for every sentence φ, either the sentence or its negation is in $Th(\Gamma)$. Thus the characteristic function χ of $Th(\Gamma)$ can be defined in terms of f as follows. First, suppose that the Gödel numbers of the collection of $\mathscr{L}$-sentences is precisely $\mathbb{N}$. Then for an $\mathscr{L}$-sentence ϕ with Gödel number n, we let

$$\chi(n) = \begin{cases} 1, & \text{if } \phi \text{ is in the range of } f \\ 0, & \text{if } \neg\phi \text{ is in the range of } f. \end{cases}$$

□

We now give an example of a computably axiomatizable, complete theory, namely the theory of dense linear orders without endpoints.

The theory of *linear orders*, denoted **LO**, in the language $\{\leq\}$ has four sentences in it, which state that the relation $\leq$ is reflexive, antisymmetric, transitive and that the order is total. The last statement for *total order* is the following:

$$(\forall x)(\forall y)(x \leq y \vee y \leq x).$$

Not that this theory does not rule out the possibility that two points in the relation $x \leq y$ are actually equal. Since we include equality in every language of predicate logic, we can also define the strict order $x < y$ to be $x \leq y \ \& \ x \neq y$.

Next, the theory of *dense linear orders*, denoted **DLO**, is obtained from the theory of linear orders by the addition of the *density property*:

$$(\forall x)(\forall y)(x < y \to (\exists z)(x < z < y))$$

Consider two more sentences that state the existence of endpoints of our linear order, REnd for right endpoint and LEnd for left endpoint:

$$(\exists x)(\forall y)(x \leq y)$$
$$(\exists x)(\forall y)(y \leq x)$$

The theory of *dense linear orders without first and last element*, denoted **DLOWE**, is the theory **DLO** with the addition of the negations of REnd and LEnd. It is this theory that we will show is decidable.

One model of the theory is quite familiar, since the rationals with the usual order is a model:

$$\langle \mathbb{Q}, \leq \rangle \models \textbf{DLOWE}$$

To show that **DLOWE** is decidable, we only need to show that it is complete, since it is clearly finitely axiomatizable (and hence computably axiomatizable). We will establish the completeness of **DLOWE** by proving a series of results.

Theorem 3.5. *Any non-empty model of* **DLOWE** *is infinite.*

Proof. Left to the reader. □

Theorem 3.6. *Any two non-trivial countable models of* **DLOWE** *are isomorphic.*

Proof. Suppose that $\mathscr{A} = \langle A, \leq_A \rangle$ and $\mathscr{B} = \langle B, \leq_B \rangle$ are two non-empty countable models of **DLOWE**. Suppose that $\langle a_i \mid i < \omega \rangle$ and $\langle b_i \mid i < \omega \rangle$ are enumerations of A and B, respectively. We define an isomorphism $h : A \to B$ by defining h in a sequence of stages.

At stage 0 of our construction, set $h(a_0) = b_0$. Suppose at stage $m > 0$, h has been defined on $\{a_{i_1}, \ldots, a_{i_k}\}$ where the elements of A are listed in increasing order under the relation $\leq_A$. Further suppose that we denote by b_{r_j} the value of $h(a_{i_j})$. Then since by hypothesis h is an isomorphism on the points on which it is defined, the elements $b_{r_1}, \ldots, b_{r_k}$ are listed in increasing order under the relation $\leq_B$.

At stage $m = 2n$ of our construction, we ensure that a_n is in the domain of h. If $h(a_n)$ has already been defined at an earlier stage, then there is nothing to do at stage $m = 2n$. Otherwise, either (i) $a_n <_A a_{i_1}$, (ii) $a_{i_k} <_A a_n$, (iii) or for some ℓ, $a_{i_\ell} <_A a_n <_A a_{i_{\ell+1}}$. Choose b_r as the element of B of least index in the enumeration $\langle b_i \mid i < \omega \rangle$ that has the same relationship to $b_{r_1}, \ldots, b_{r_k}$ that a_n has to $a_{i_1}, \ldots, a_{i_k}$. It is possible to choose such a b_r in (i) the first case because B has no left endpoint, (ii) the second case because B has no right endpoint, and (iii) the last case because the order is dense. Extend h to a_n by setting $h(a_n) = b_r$.

At stage $m = 2n + 1$ of our construction, we ensure that the point b_n is in the range of h. As above, if b_n is in the range of h, then there is nothing to do at stage $m = 2n + 1$. Otherwise, it has a unique position relative to $b_{r_1}, \dots, b_{r_k}$. As above, either it is a left endpoint, a right endpoint, or it lies strictly between b_{r_ℓ} and $b_{r_{\ell+1}}$ for some ℓ. As in the previous case, choose a_r as the element of A of least index in the enumeration of $\langle a_i \mid i < \omega \rangle$ which has the same relationship to $a_{i_1}, \dots, a_{i_k}$ as b_n has to $b_{r_1}, \dots, b_{r_k}$, and extend h to a_r by setting $h(a_r) = b_n$.

This completes the recursive definition of h. One can readily prove by induction that the domain of h is A, the range of h is B, and that h is an isomorphism. □

This property of **DLOWE** is an example of a more general property.

Definition 3.7. *A theory Γ is κ-categorical for some infinite cardinal κ if and only if every two models of Γ of cardinality κ are isomorphic.*

Corollary 3.8. *The theory* **DLOWE** *is $\aleph_0$-categorical.*

The key result here is that theories that are categorical in some power and have only infinite models are also complete.

Theorem 3.9 (Los-Vaught Test). *Suppose that Γ is a theory of cardinality κ with no finite models. If Γ is λ-categorical for some (infinite) $\lambda \geq \kappa$, then Γ is complete.*

Proof. Suppose by way of contradiction that Γ is not complete, but is λ-categorical for some $\lambda \geq \kappa$. Then there is a sentence φ in the language of Γ such that neither $\Gamma \vdash \varphi$ nor $\Gamma \vdash \neg\varphi$. Let $\Gamma_1 = \Gamma \cup \{\varphi\}$ and $\Gamma_2 = \Gamma \cup \{\neg\varphi\}$. Since Γ is consistent, so are Γ_1 and Γ_2. By the Completeness Theorem, each of these theories has a model. Since both of these models are also models of Γ, by hypothesis, they must be infinite. Therefore by the Löwenheim-Skolem Theorem, they each have models of cardinality λ. Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be models of Γ_1 and Γ_2 of cardinality λ. Both $\mathcal{N}_1$ and $\mathcal{N}_2$ are models of Γ of power λ, so they are isomorphic. By recursion on the definition of $\models$, one can prove that they model the same set of sentences. However $\mathcal{N}_1$ is a model of φ while $\mathcal{N}_2$ is a model of $\neg\varphi$. This contradiction proves that Γ is complete. □

Corollary 3.10. *The theory* **DLOWE** *is complete and hence decidable.*

Other examples of decidable theories are:

(1) Presburger arithmetic (arithmetic without multiplication);
(2) arithmetic with only multiplication and no addition;
(3) the theory of real closed fields (fields F in which every polynomial of odd degree has a root in F);
(4) the first-order theory of algebraically closed fields of a fixed characteristic; and
(5) the first-order theory of Euclidean geometry.

4. Gödel's Incompleteness Theorems

We now turn to undecidable theories. We will prove Gödel's first incompleteness theorem (G1), which states that any computably axiomatizable theory that contains elementary arithmetic (such as Peano Arithmetic and ZFC) is incomplete and hence undecidable. Gödel's second incompleteness theorem (G2) states that any computably axiomatizable theory that contains a sufficient amount of arithmetic cannot prove its own consistency. We will show this holds for Peano Arithmetic.

Just how much arithmetic is sufficient to prove the incompleteness theorems? For G1, it suffices to satisfy the theory of arithmetic in the language $\{+, \times, S, 0\}$ known as Robinson's Q, given by the following axioms:

(Q_1) $\neg(\exists x)S(x) = 0$
(Q_2) $(\forall x)(\forall y)(S(x) = S(y) \rightarrow x = y)$
(Q_3) $(\forall x)(x \neq 0 \rightarrow (\exists y)x = S(y))$
(Q_4) $(\forall x)(x + 0 = x)$
(Q_5) $(\forall x)(\forall y)(x + S(y) = S(x + y))$
(Q_6) $(\forall x)(x \times 0 = 0)$
(Q_7) $(\forall x)(\forall y)(x \times S(y) = (x \times y) + x)$

Note that Q is finitely axiomatizable and hence is computably axiomatizable.

To carry out the proof of $G2$, a stronger theory of arithmetic is necessary (while there is a version of G2 for Q, it is generally accepted that Q lacks the resources to recognize that the statement asserting its consistency really is its own consistency statement). Such a theory can be obtained by expanding our language to include symbols for every primitive function, adding axioms to Q that define every primitive recursive function, and adding mathematical induction for quantifier-free formulas (that is, for every quantifier-free formula ϕ in our language, we include as an axiom the sentence $(\phi(0) \,\&\, (\forall n)(\phi(n) \rightarrow \phi(S(n))) \rightarrow (\forall n)\phi(n))$. The resulting theory is known as *primitive recursive arithmetic*, which one can verify is computably axiomatizable.

We will restrict our attention to Peano Arithmetic (hereafter, *PA*), which is obtained by adding to Q the schema of mathematical induction for all formulas in the language of arithmetic. *PA* is also computably axiomatizable.

In order to prove Gödel's theorems, we have to represent the notion of provability *within PA*. To do so requires us to code much of the meta-theory of *PA* inside of *PA*. We begin by coding the natural numbers in the language of arithmetic as follows:

number	name	abbreviation
0	0	$\underline{0}$
1	S(0)	$\underline{1}$
2	S(S(0))	$\underline{2}$
$\vdots$	$\vdots$	$\vdots$

Hereafter, the name of each $n \in \mathbb{N}$ in *PA* will be written $\underline{n}$.

Next, we represent functions in *PA*.

Definition 4.1. *A function $f : \mathbb{N}^k \rightarrow \mathbb{N}$ is* definable in *PA by a formula ϕ if the following equivalence holds: For every $u_1, \dots, u_k, v \in \mathbb{N}$*

$$PA \vdash \phi(\underline{u_1}, \dots, \underline{u_k}, \underline{v}) \text{ if and only if } f(u_1, \dots, u_k) = v.$$

A relation is definable *if its characteristic function is definable.*

We now code sequences of natural numbers in a computable and definable fashion. Although we could use a coding like the one using prime numbers as in Section 1, we will use an alternative coding.

Lemma 4.2. *The following relations are both computable and definable in PA.*

(1) $x \leq y$;
(2) $rem(x, y) = z$ *where z is the remainder x is divided by y;*
(3) $Code(x, y, z) = rem(x, 1 + (z + 1)y)$.

Proof. (1) $(x \leq y)$: The formula that defines this relation in *PA* is

$$x \leq y \text{ if and only if } (\exists z)(x + z = y).$$

To see that the relation is computable, notice that since $x \leq y$ if and only if $x \dot{-} y = 0$, the characteristic function of the relation $x \leq y$ is

$$\chi_{\leq}(x, y) = 1 \dot{-} (x \dot{-} y)$$

(2) $(rem(x, y))$: The formula that defines this relation in *PA* is

$$rem(x, y) = z \text{ if and only if } (\exists q)((x = yq + z) \& (0 \leq z < y)).$$

To see that the relation is computable, notice that the desired value of q is the least q so that $y(q+1) > x$. That is, this is the least $q \leq x$ such that it is not the case that $y(q+1) \leq x$, or equivalently, the least $q \leq x$ such that $\chi_{\leq}(y(q+1), x) = 0$, which can be found by bounded search. If such a q exists, then we check to see if $z = x - yq$, which can be done computably.

(3) $(Code(x, y, z))$: The definition of *Code* in *PA* is given above. Since addition, multiplication and *rem* are all computable, so is *Code*.

□

Theorem 4.3. *For any sequence $k_1, k_2, \ldots, k_n$ of natural numbers, there exist natural numbers a, b such that $Code(a, b, 0) = n$ and* $\mathrm{Code}(a, b, i) = k_i$ *for $i = 1, 2, \ldots, n$.*

Proof. We prove this using the Chinese Remainder Theorem, which is as follows: Let $m_1, \ldots, m_n$ be pairwise relatively prime (i.e., $gcd(m_i, m_j) = 1$ for $i \neq j$). Then for any $a_1, \ldots, a_n \in \mathbb{N}$, the system of equations

$$\begin{aligned} x &\equiv a_1 \mod m_1 \\ &\vdots \\ x &\equiv a_n \mod m_n \end{aligned}$$

has a unique solution modulo $M = m_1 \cdot \ldots \cdot m_n$. To use the Chinese Remainder Theorem to prove the theorem, let $s = \max\{n, k_1, \ldots, k_n\}$ and set $b = s!$. We claim that the numbers

$$s! + 1, 2s! + 1, \ldots, (n+1)s! + 1$$

are pairwise relatively prime. Suppose not. Then there are $c, d \in \mathbb{N}$ such that $0 \leq c < d \leq n+1$ and $p \in \mathbb{N}$ such that $p \mid (cs! + 1)$ and $p \mid (ds! + 1)$. Note that $p > s$, since $p \leq s$ implies that $rem(cs! + 1, p) = 1$. $p \mid (cs! + 1)$ and $p \mid (ds! + 1)$ together imply that

$$p \mid ((ds! + 1) - (cs! + 1)) = (d - c)s!.$$

However, $p \nmid s!$, since $p \mid s!$ implies $p \nmid (cs! + 1)$. It follows that $p \mid (d - c)$. But $d - c < n \leq s$, so $p \leq s$, which contradicts our earlier statement. By the Chinese Remainder Theorem, there is a unique solution x to

$$\begin{aligned} x &\equiv n \quad \mod s! + 1 \\ x &\equiv k_1 \quad \mod 2s! + 1 \\ &\vdots \\ x &\equiv k_n \quad \mod (n+1)s! + 1 \end{aligned}$$

modulo $M = \prod_{i=1}^{n+1} is! + 1$. Let $a = x$.

We now check that a and b are the desired values. First,

$$\begin{aligned} Code(a, b, 0) &= rem(a, b + 1) \\ &= rem(a, s! + 1) = n. \end{aligned}$$

For $i = 1, \dots, n$,

$$\begin{aligned} Code(a,b,i) &= rem(a,(i+1)b+1) \\ &= rem(a,(i+1)s!+1) = k_i. \end{aligned}$$

□

Hereafter, let us fix a Gödel number of the symbols in the language of arithmetic:

symbol	code	symbol	code
(	1	$=$	8
)	2	$\exists$	9
&	3	$\forall$	10
$\neg$	4	$+$	11
$\vee$	5	$\times$	12
$\rightarrow$	6	S	13
$\leftrightarrow$	7	v_i	$14+i$

Using the function *Code*, we now code as natural numbers the following objects: $\mathscr{L}$-formulas, $\mathscr{L}$-sentences, sequences of $\mathscr{L}$-sentences, and proofs. First, we note that a pair of natural numbers (a,b) can be coded as a single natural number, denoted $\langle a,b\rangle$, by means of the following function:

$$\langle x,y\rangle = \frac{1}{2}(x+y)(x+y+1)+y.$$

Now, given an $\mathscr{L}$-formula ϕ, if a,b are natural numbers such that $Code(a,b,z)$ outputs the Gödel numbers of the symbols that make up ϕ, then we will set $\ulcorner\phi\urcorner = \langle a,b\rangle$. That is, $\ulcorner\phi\urcorner$ will denote the Gödel number of ϕ.

Next, if $\phi_1,\dots,\phi_k$ is a sequence of $\mathscr{L}$-formulas, then n is the Gödel number of this sequence if $n = \langle c,d\rangle$ and

$$\begin{aligned} Code(c,d,0) &= k \\ Code(c,d,i) &= \ulcorner\phi_i\urcorner \end{aligned}$$

for $i = 1,\dots,k$.

Recall that a proof $\Sigma \vdash \phi$ is a finite sequence of $\mathscr{L}$-formulas $\psi_1,\dots,\psi_n,\phi$, where each $\psi_i \in \Sigma$ or follows from some subcollection of $\{\psi_1,\dots,\psi_{i-1}\}$ by one of the rules of inference. We will make use of the following, which can be proved by showing that each of the rules of inference in the predicate calculus can effectively verified to hold.

Lemma 4.4. *There is a computable, definable predicate $Proof(n,m) \subseteq \mathbb{N}^2$ (represented by an $\mathscr{L}$-formula that we will also denote $Proof(n,m)$ such that $PA \vdash Proof(\underline{n},\underline{m})$ if and only if $\underline{m}$ is the code of a sequence of $\mathscr{L}$-sentences $\psi_1,\dots,\psi_n,\phi$ such that $\{\psi_1,\dots,\psi_n\} \vdash \phi$ and $\ulcorner\phi\urcorner = n$.*

We will make use of the following in the proof of Gödel's first incompleteness theorem.

Theorem 4.5. *The partial computable functions are exactly the functions definable in PA.*

We prove two auxiliary lemmas.

Lemma 4.6. $f : \mathbb{N}^k \to \mathbb{N}$ *is partial computable if and only if*

$$Graph(f) = \{(x_1,\dots,x_k,y) : f(x_1,\dots,x_k) = y\}$$

is a computably enumerable set.

Proof. ($\Rightarrow$) If f is partial computable, then we can define an effective procedure that enumerates a tuple $(n_1,\dots,n_k,m)$ into *Graph*(f) whenever we see $f(n_1,\dots,n_k)\downarrow = m$.

($\Leftarrow$) To compute $f(n_1,\dots,n_k)$, enumerate *Graph*(f) until we see some tuple $(n_1,\dots,n_k,m)$ that belongs to *Graph*(f). If no such tuple exists, then we will wait forever and hence $f(n_1,\dots,n_k)\uparrow$. □

Lemma 4.7. *$S \subseteq \mathbb{N}^k$ is computably enumerable if and only if there is a computable relation $R \subseteq \mathbb{N}^{k+1}$ such that*

$$S = \{(n_1,\dots,n_k) : \exists z\, R(n_1,\dots,n_k,z)\}.$$

We will omit the proof this lemma.

Proof of Theorem 4. ($\Leftarrow$): Suppose that $f : \mathbb{N}^k \to \mathbb{N}$ is definable in *PA*. Then there is some $\mathscr{L}$-formula ϕ_f such that for all $u_1,\dots,u_k,v \in \mathbb{N}$,

$$\begin{aligned} PA \vdash \phi(\underline{u_1},\dots,\underline{u_k},\underline{v}) &\Leftrightarrow f(u_1,\dots,u_k) = v \\ &\Leftrightarrow (u_1,\dots,u_k,v) \in \mathit{Graph}(f). \end{aligned}$$

However, $PA \vdash \phi(\underline{u_1},\dots,\underline{u_k},\underline{v})$ if and only if there is some finite $X \subseteq PA$ such that $X \vdash \phi(\underline{u_1},\dots,\underline{u_k},\underline{v})$. Equivalently, there is some m such that *Proof*$(m, \ulcorner\phi_f(\underline{u_1},\dots,\underline{u_k},\underline{v})\urcorner)$ holds (where m codes a sequence $\psi_1,\dots,\psi_j,\phi_f(\underline{u_1},\dots,\underline{u_k},\underline{v})$ where $\{\psi_1,\dots,\psi_j\} \vdash \phi_f(\underline{u_1},\dots,\underline{u_k},\underline{v})$). It follows that

$$\mathit{Graph}(f) = \{(u_1,\dots,u_k,v) : \exists m\, R(u_1,\dots,u_k,v,m)\}$$

where $R(u_1,\dots,u_k,v,m)$ is the computable relation *Proof*$(m, \ulcorner\phi_f(\underline{u_1},\dots,\underline{u_k},\underline{v})\urcorner)$. By Lemma 4.7, *Graph*$(f)$ is computably enumerable, and so by Lemma 4.6, f is partial computable.

($\Rightarrow$) We show all partial computable functions are definable in *PA* by induction. It is not hard to show that the initial functions are definable.

(1) (Constant function): The defining formula $\phi_c(x,y)$ for the constant function $c(x) = 0$ is
$$(y = \underline{0})\ \&\ (x = x).$$
(2) (Projective functions): The defining formula $\phi_{p^k_j}(x_1,\dots,x_k,y)$ for the projection function $p^k_j(x_1,\dots,x_k) = x_j$ is
$$(y = x_j)\ \&\ (x_1 = x_1)\ \&\ \dots\ \&\ (x_k = x_k).$$
(3) (Successor function): The defining formula $\phi_S(x,y)$ for the successor function $S(x) = x+1$ is $y = S(x)$.

For the induction step of the proof, we must show that the set of *PA*-definable functions is closed under the production rules for the set of partial computable functions.

(4) (Composition): Suppose that f and $g_1,g_2,\dots,g_k$ are definable in *PA* by the formulas ϕ_f and $\phi_{g_1},\dots,\phi_{g_k}$, respectively and that h is the function $h(\vec{x}) = f(g_1(\vec{x}),\dots,g_k(\vec{x}))$. Then the defining formula $\phi_h(\vec{x},y,z)$ for h is
$$(\exists y_1)(\exists y_2)\dots(\exists y_k)(\phi_{g_1}(\vec{x},y_1)\ \&\ \dots\ \&\ \phi_{g_k}(\vec{x},y_k)\ \&\ \phi_f(y_1,\dots,y_k,z)).$$
(5) (Primitive Recursion): Suppose that f and g are definable in *PA* by ϕ_f and ϕ_g, respectively, and that h is the function defined by recursion with
$$\begin{aligned} h(\vec{x},0) &= f(\vec{x}) \\ h(\vec{x},y+1) &= g(\vec{x},y,h(\vec{x},y)). \end{aligned}$$

To define h, we will use a pair of numbers a and b that code up the sequence

$$h(\vec{x},0), h(\vec{x},1), \dots, h(\vec{x},y),$$

via the function *Code*, where $h(\vec{x},0) = f(\vec{x})$ and $h(\vec{x},n+1) = g(\vec{x},n,h(\vec{x},n))$ for every $n < y$. Thus the defining formula $\phi_h(\vec{x},y,z)$ for h is

$$\begin{aligned}(\exists a)(\exists b)(&\phi_f(\vec{x}, Code(a,b,1))\\ &\&\ (\forall i < y)(\phi_g(\vec{x}, i, Code(a,b,i+1), Code(a,b,i+2))\\ &\&\ (z = Code(a,b,y+1))).\end{aligned}$$

(6) (Unbounded search): Left to the reader.

Thus the class of functions definable in *PA* includes all the initial functions and is closed under the production rules. Therefore every partial computable function is definable in *PA*. □

The following is a key ingredient of the proofs of the incompleteness theorems.

Lemma 4.8 (Diagonal lemma). *Let $\phi(x)$ be an $\mathscr{L}$-formula with one free variable. Then there is an $\mathscr{L}$-sentence ψ such that*

$$PA \vdash \psi \leftrightarrow \phi(\ulcorner \psi \urcorner).$$

Proof. First, observe that the operation that, given inputs $n \in \mathbb{N}$ and a formula $A(x)$ with one free variable, outputs $A(n)$ is purely mechanical. Similarly, the operation that, given inputs $n \in \mathbb{N}$ and the Gödel number of such a formula $A(x)$, outputs $\ulcorner A(\underline{n}) \urcorner$, the Gödel number of $A(\underline{n})$, is also purely mechanical, and hence is intuitively computable.

Let $subst(\ulcorner A(x) \urcorner, n) = \ulcorner A(\underline{n}) \urcorner$ be this function, which is partial computable (since it may receive as input some number that is not the Gödel number of any formula, or the Gödel number of a formula with more than one free variable, etc.). By Theorem 4, there is a formula $S(x,y,z)$ that defines *subst* in *PA*.

The key observation to make here is that for any $\mathscr{L}$-formula $A(x)$ with one free variable, we can consider the value $subst(\ulcorner A(x) \urcorner, \ulcorner A(x) \urcorner)$. Given $\phi(x)$ as above, let $\theta(x)$ be the formula

$$(\exists y)(\phi(y)\ \&\ S(x,x,y)).$$

That is, $\theta(x)$ says "There is some y satisfying ϕ that is obtained by substituting x into the formula with Gödel number x." Let $k = \ulcorner \theta(x) \urcorner$ and consider the sentence $\theta(\underline{k})$, which has Gödel number $\ulcorner \theta(\underline{k}) \urcorner$. Call this sentence ψ.

Unpacking the sentence $\theta(\underline{k})$, we have

$$(\exists y)(\phi(y)\ \&\ S(\underline{k},\underline{k},y))$$

Moreover, since $subst(k,k) = \ulcorner \theta(\underline{k}) \urcorner$, one can show that

$$PA \vdash (\forall y)(S(\underline{k},\underline{k},y) \leftrightarrow y = \ulcorner \theta(\underline{k}) \urcorner). \tag{1}$$

Then by definition of ψ

$$PA \vdash \psi \leftrightarrow (\exists y)(\phi(y)\ \&\ S(\underline{k},\underline{k},y)).$$

However, by Equation 1,

$$PA \vdash \psi \leftrightarrow (\exists y)(\phi(y)\ \&\ y = y = \ulcorner \theta(\underline{k}) \urcorner).$$

Equivalently,

$$PA \vdash \psi \leftrightarrow \phi(\ulcorner \theta(\underline{k}) \urcorner),$$

which can be rewritten as

$$PA \vdash \psi \leftrightarrow \phi(\ulcorner \psi \urcorner).$$

□

One last necessary result that we will state without proof is the following.

Lemma 4.9. *Given an $\mathscr{L}$-sentence ψ of the form $(\exists x)\phi(x)$, if ψ is true in the standard model of arithmetic, then ψ is provable in PA. That is,*

$$\mathbb{N} \models (\exists x)\phi(x) \Leftrightarrow PA \vdash (\exists x)\phi(x).$$

Theorem 4.10 (Gödel's First Incompleteness Theorem)**.** *If PA is consistent, then PA is not complete. That is, there is a sentence ψ_G true in the standard model of arithmetic such that $PA \nvdash \psi_G$ and $PA \nvdash \neg\psi_G$.*

Proof. We apply the Diagonal Lemma to the formula $\neg(\exists x)Proof(x,y)$, which we will abbreviate as $\neg Prov(y)$ (Informally, this sentence asserts that there is no proof of the sentence with Gödel number y). Thus, there is some $\mathscr{L}$-sentence ψ_G (the "Gödel sentence") such that

$$PA \vdash \psi_G \leftrightarrow \neg Prov(\ulcorner \psi_G \urcorner). \tag{2}$$

That is, ψ_G is equivalent to the $\mathscr{L}$-sentence that asserts that ψ_G is not provable.

We now show that if PA is consistent, then $PA \nvdash \psi_G$ and $PA \nvdash \neg\psi_G$. In fact, we will show that $PA \vdash \psi_G$ if and only if $PA \vdash \neg\psi_G$, which is clearly impossible if PA is consistent and from which the desired conclusion follows.

Observe that $PA \vdash \psi_G$ if and only if $(\exists n)Proof(n, \ulcorner \psi_G \urcorner)$ holds, which is equivalent to $PA \vdash (\exists n)Proof(n, \ulcorner \psi_G \urcorner)$ by Lemma 4.9. This is equivalent to $PA \vdash Prov(\ulcorner \psi_G \urcorner)$, which is equivalent to $PA \vdash \neg\psi_G$ by (2).

Lastly, if PA is consistent, then $\mathbb{N} \models \psi_G$. Suppose instead that $\mathbb{N} \models \neg\psi_G$. Since $\neg\psi_G$ is equivalent to $(\exists n)Proof(n, \ulcorner \psi_G \urcorner)$, it follows that $\mathbb{N} \models (\exists n)Proof(n, \ulcorner \psi_G \urcorner)$. But then by Lemma 4.9, it follows that $PA \vdash (\exists n)Proof(n, \ulcorner \psi_G \urcorner)$, i.e., $PA \vdash Prov(\ulcorner \psi_G \urcorner)$. But this implies that $PA \vdash \neg\psi_G$, which we have shown is impossible.

□

We now turn to Gödel's second incompleteness theorem. Informally, Gödel's second theorem states that if PA is consistent, then it cannot prove that it is consistent. To express this formally, we need to formalize that statement that PA is consistent *within PA*. The standard way to do this by the formula $\neg Prov(\ulcorner \underline{0} = \underline{1} \urcorner)$.

Theorem 4.11 (Godel's Second Incompleteness Theorem)**.** *If PA is consistent, then*

$$PA \nvdash \neg Prov(\ulcorner \underline{0} = \underline{1} \urcorner).$$

To prove G2, one can show (with a considerable amount of work) that the proof of G1 can be carried out entirely *within PA*. We proved above that if PA is consistent, then $PA \nvdash \psi_G$; within PA, this yields

$$PA \vdash \neg Prov(\ulcorner \underline{0} = \underline{1} \urcorner) \rightarrow \neg Prov(\ulcorner \psi_G \urcorner).$$

By G1, we have

$$PA \vdash \psi_G \leftrightarrow \neg Prov(\ulcorner \psi_G \urcorner).$$

It follows from the previous two statements that

$$PA \vdash \neg Prov(\ulcorner \underline{0} = \underline{1} \urcorner) \rightarrow \psi_G.$$

Thus if $PA \vdash \neg Prov(\ulcorner \underline{0} = \underline{1} \urcorner)$, it would follow that $PA \vdash \psi_G$, which is impossible by G1. Hence $PA \nvdash \neg Prov(\ulcorner \underline{0} = \underline{1} \urcorner)$.

Let us take a more general approach to proving G2. Let $B(x)$ be an $\mathscr{L}$-formula with one free variable. The following three conditions are referred to as *derivability conditions*:

(D1) If $PA \vdash \phi$, then $PA \vdash B(\ulcorner \phi \urcorner)$.
(D2) $PA \vdash B(\ulcorner \phi \to \psi \urcorner) \to (B(\ulcorner \phi \urcorner) \to B(\ulcorner \psi \urcorner))$.
(D3) $PA \vdash B(\ulcorner \phi \urcorner) \to B(\ulcorner B(\ulcorner \phi \urcorner) \urcorner)$.

With some effort, one can prove that if we let $B(x)$ be the formula $Prov(x)$, then (D1)-(D3) hold.

We can now formulate an abstract version of G2:

Theorem 4.12 (Abstract G2). *If $B(x)$ satisfies (D1)-(D3), then $PA \nvdash \neg B(\ulcorner \underline{0} = \underline{1} \urcorner)$.*

To prove this theorem, we will first prove the following, which tells us that *PA* can only prove that soundness for sentences that it can prove to hold.

Theorem 4.13 (Löb's Theorem). *If $B(x)$ satisfies (D1)-(D3), then for any $\mathscr{L}$-sentence ϕ, if $PA \vdash B(\ulcorner \phi \urcorner) \to \phi$, then $PA \vdash \phi$.*

Proof. Suppose that $B(x)$ satisfies (D1)-(D3) and that

$$PA \vdash B(\ulcorner \phi \urcorner) \to \phi. \tag{3}$$

Let $\theta(y)$ be the formula $B(y) \to \phi$. By the Diagonal Lemma there is an $\mathscr{L}$-sentence ψ such that

$$PA \vdash \psi \leftrightarrow (B(\ulcorner \psi \urcorner) \to \phi) \tag{4}$$

and hence

$$PA \vdash \psi \to (B(\ulcorner \psi \urcorner) \to \phi). \tag{5}$$

From (5) and (D1) it follows that

$$PA \vdash B(\ulcorner \psi \to (B(\ulcorner \psi \urcorner) \to \phi) \urcorner). \tag{6}$$

By (D2)

$$PA \vdash B(\ulcorner \psi \to (B(\ulcorner \psi \urcorner) \to \phi) \urcorner) \to (B(\ulcorner \psi \urcorner) \to B(\ulcorner B(\ulcorner \psi \urcorner) \to \phi \urcorner)) \tag{7}$$

Then by applying modus ponens to (6) and (7), we have

$$PA \vdash B(\ulcorner \psi \urcorner) \to B(\ulcorner B(\ulcorner \psi \urcorner) \to \phi \urcorner). \tag{8}$$

Again by (D2),

$$PA \vdash B(\ulcorner B(\ulcorner \psi \urcorner) \to \phi \urcorner) \to (B(\ulcorner B(\ulcorner \psi \urcorner) \urcorner) \to B(\ulcorner \phi \urcorner)). \tag{9}$$

From (8) and (9) we have

$$PA \vdash B(\ulcorner \psi \urcorner) \to (B(\ulcorner B(\ulcorner \psi \urcorner) \urcorner) \to B(\ulcorner \phi \urcorner)). \tag{10}$$

By (D3),

$$PA \vdash B(\ulcorner \psi \urcorner) \to B(\ulcorner B(\ulcorner \psi \urcorner) \urcorner), \tag{11}$$

and so from (10) and (11) we can conclude

$$PA \vdash B(\ulcorner \psi \urcorner) \to B(\ulcorner \phi \urcorner). \tag{12}$$

By (3) and (12) we have

$$PA \vdash B(\ulcorner \psi \urcorner) \to \phi. \tag{13}$$

From (4) and (13) it follows that

$$PA \vdash \psi. \tag{14}$$

From (D1) we can infer from (14) that

$$PA \vdash B(\ulcorner \psi \urcorner). \tag{15}$$

Applying modus ponens to (13) and (15) gives the desired conclusion

$$PA \vdash \phi.$$

□

Proof of Abstract G2. Suppose $B(x)$ satisfies (D1)-(D3) and that $PA \vdash \neg B(\ulcorner \underline{0} = \underline{1} \urcorner)$. Then

$$PA \vdash B(\ulcorner \underline{0} = \underline{1} \urcorner) \rightarrow \phi$$

for any $\mathscr{L}$-sentence ϕ. In particular,

$$PA \vdash B(\ulcorner \underline{0} = \underline{1} \urcorner) \rightarrow \underline{0} = \underline{1}.$$

Then by Löb's Theorem, it follows that $PA \vdash \underline{0} = \underline{1}$, which contradicts our assumption that *PA* is consistent. Thus $PA \nvdash \neg B(\ulcorner \underline{0} = \underline{1} \urcorner)$. □

FOUNDATIONS OF MATHEMATICS
CHAPTER 7: ALGORITHMIC RANDOMNESS

1. Introduction

In this chapter, we will briefly the discuss the basics of the theory of algorithmically random strings. Although this is not a topic typically covered in an introduction to the foundations of mathematics, the theory of algorithmic randomness provides interesting examples of the phenomena of undecidability and incompleteness that we considered in the previous two chapters.

How exactly do we define a finite binary string to be random? One possibility is to take such a string to be random if it is produced by a paradigmatically random process such as the tosses of a fair coin. For some purposes, such a definition is sufficient. For our purposes, however, we would like a definition that picks out a fixed collection of binary strings as the random ones (although this is not quite what the theory of algorithmically random finite strings delivers for us).

As a first step, consider the following binary strings of length 50:

(1) 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000
(2) 01010 10101 01010 10101 01010 10101 01010 10101 01010 10101
(3) 10100 00011 01010 00110 10110 10001 11110 00011 11101 00011
(4) 00100 10000 11111 10110 10101 00010 00100 00101 10100 01100
(5) 01001 00101 10101 11111 11010 10100 11110 01111 11111 10010

Although it is possible to obtain each of these strings by tossing an unbiased coin (where we take 0 to stand for heads and 1 to stand for tails), it is *highly* unlikely that we would produce (1) and (2) in this way. In fact, (1) and (2) just do not appear to be the sort of strings produced by the tosses of an unbiased coin. It is less clear whether (3)-(5) can be thus produced. For instance, (3) has the same number of 0s and 1s, and since most strings of length 50 have close to the same number of 0s and 1s, we might surmise that (3) was produced by tossing a fair coin. However, (3) encodes the parity of the number of words of each of the 50 states in alphabetical order (e.g. 'Alabama' has an odd number of letters and thus corresponds to a '0'). Although the digits of (4) are less evenly distributed than those of (3) (29 0s and 21 1s), this is not inconsistent a random origin. However, once we learn that (4) consists of the first 50 bits of the binary expansion of π, we would still be surprised if we were to produce this string by coin tossing. Lastly, (5) is slightly more biased than (3) or (4), but this was (purportedly) produced by encoding some atmospheric noise as a binary string (at least according to random.org).

What is the upshot of this discussion? One is that strings that appear to be random such as (3) and (4) are no longer thought to be random when they can be given some simpler description. This is one key idea behind the definition of an algorithmically random string: random strings are difficult to describe. Here we must exercise some caution: here we consider a description of a string to be one that allows us to reconstruct the string unambiguously, like a blueprint for reproducing the string. The other key idea behind the definition of an algorithmically random string is that the descriptions we consider are computational, given in some way by a Turing machine.

2. Kolmogorov Complexity

We now make the above ideas precise by defining the Kolmogorov complexity of string, which allows us to define the complexity of effectively reproducing that string. We begin with an example.

Example 2.1. We define a Turing machine $M : 2^{<\omega} \to 2^{<\omega}$ that gives short descriptions of strings of the form 1^n for $n \in \mathbb{N}$. Given an input string $\sigma \in 2^{<\omega}$, M interprets this as some $n \in \mathbb{N}$ written in (reverse) binary and outputs n 1s. Note that if $n = 2^k$ for some k, then the input string representing n will have length $k+1$ and will yield as output a string of length 2^k, a considerable gain. Thus, we can think of the input string as a compressed version of the output string.

The previous definition suggests the following definition.

Definition 2.2. *Let $M : 2^{<\omega} \to 2^{<\omega}$ be a Turing machine and let $\sigma \in 2^{<\omega}$. We define the* Kolmogorov complexity of σ relative to M *to be*

$$C_M(\sigma) = \min\{|\tau| : M(\tau)\downarrow = \sigma\},$$

where $|\tau|$ is the length of τ. Moreover, if $\sigma \notin ran(M)$, then we set $C_M(\sigma) = \infty$.

If $M(\tau) = \sigma$, we can think of τ as an M-description of σ. Then $C_M(\sigma)$ is the length of the shortest M-description of σ.

Example 2.3. If, as in the previous example, $M(\sigma_n) = 1^n$, where σ_n is the binary representation of $n \in \mathbb{N}$, then

$$C_M(1^n) = \lfloor \log_2(n) \rfloor + 1$$

(and $C_M(\sigma) = \infty$ if $\sigma \neq 1^n$ for every $n \in \mathbb{N}$).

One might reasonably worry here that this measure of complexity is completely dependent on our choice of underlying machine. This worry is not unjustified. However, this worry can be mitigated if we define complexity in terms of a universal Turing machine.

Let $(M_e)_{e\in\mathbb{N}}$ be a computable enumeration of all Turing machines (effectively coded in some way). Then we define $U : 2^{<\omega} \to 2^{<\omega}$ by

$$U(1^e 0\tau) \simeq M_e(\tau)$$

for every $e \in \mathbb{N}$ and $\sigma \in 2^{<\omega}$.

Definition 2.4. *The* Kolmogorov complexity *of σ is defined to be*

$$C(\sigma) := C_U(\sigma) = \min\{|\tau| : U(\tau)\downarrow = \sigma\}.$$

To what extent does this definition address the concern with machine dependence? The following theorem, first proved by Kolmogorov, provides an answer:

Theorem 2.5 (Invariance Theorem). *For any Turing machine $M : 2^{<\omega} \to 2^{<\omega}$, there is some $c \in \mathbb{N}$ such that*

$$C(\sigma) \leq C_M(\sigma) + c$$

for every $\sigma \in 2^{<\omega}$.

Proof. Given $M : 2^{<\omega} \to 2^{<\omega}$, let e be such that $M_e = M$. Then for all $\tau \in 2^{<\omega}$, $U(1^e 0\tau) \simeq M_e(\tau)$. Given $\sigma \in 2^{<\omega}$, let σ^* be the shortest string such that $M_e(\sigma^*_M) = \sigma$. Then $C_M(\sigma) = |\sigma^*|$. Since $U(1^e 0\sigma^*) = M_e(\sigma^*) = \sigma$, it follows that

$$C(\sigma) \leq |1^e 0\sigma^*| = |\sigma^*| + e + 1 = C_M(\sigma) + e + 1.$$

The conclusion follows by setting $c = e + 1$. □

We now find a simple upper bound for C.

Proposition 2.6. *There is some $c \in \mathbb{N}$ such that*

$$C(\sigma) \leq |\sigma| + c$$

for every $\sigma \in 2^{<\omega}$.

Proof. Let $M_{id} : 2^{<\omega} \to 2^{<\omega}$ be the identity machine, which satisfies $M(\sigma) = \sigma$ for every $\sigma \in 2^{<\omega}$. Then by the Invariance Theorem, there is some $c \in \mathbb{N}$ such that

$$C(\sigma) \leq C_{M_{id}}(\sigma) + c = |\sigma| + c.$$

□

3. Incompressible strings

We now formalize the notion of an incompressible string, which will serve as the basis of our definition of random strings.

Definition 3.1. *Let $c \in \mathbb{N}$. We say that $\sigma \in 2^{<\omega}$ is* c-incompressible *if*

$$C(\sigma) \geq |\sigma| - c.$$

Do c-incompressible strings exist? Let us consider the case that $c = 0$. Note that $\sigma \in 2^{<\omega}$ is 0-incompressible if for every $\tau \in 2^{<\omega}$ such that $U(\tau)\!\downarrow = \sigma$, it follows that $|\tau| \geq |\sigma|$. We claim that there is at least one 0-incompressible string of each length. Let us consider strings of length n. There are

$$1 + 2 + \ldots + 2^{n-1} = 2^n - 1$$

strings of length $< n$. Hence, in the worst case, if each string of length $< n$ is mapped by U to a unique string of length n, then one string of length n remains and is hence 0-incompressible. We now generalize this line of reasoning to prove the following.

Proposition 3.2. *For each $c \in \mathbb{N}$ and $n \in \mathbb{N}$, there are at least $2^n - (2^{n-c} - 1)$ c-incompressible strings of length n.*

Proof. There are

$$1 + 2 + \ldots + 2^{(n-c)-1} = 2^{n-c} - 1$$

strings of length $< n - c$. Thus the number of strings of length n to which U maps a string of length $< n - c$ is at most $2^{n-c} - 1$. Hence, the number of strings of length n to which U maps no string of length $< n - c$ is at least $2^n - 2^{n-c} - 1$. Since all such strings are c-incompressible, the conclusion follows. □

Note that $2^n - (2^{n-c} - 1) = 2^n(1 - 2^{-c}) + 1 > 2^n(1 - 2^{-c})$. From this we can conclude that for each $n \in \mathbb{N}$

— at least 1/2 of the strings of length n are 1-incompressible;
— at least 3/4 of the strings of length n are 2-incompressible;
— at least 7/8 of the strings of length n are 3-incompressible;
— and so on.

Thus, if we consider, say, strings of length 100, 99.9% of them are 10-incompressible; that is at least 1023/1024 of them have complexity greater than 90. It is thus quite easy to produce such a string: toss a fair coin 100 times, and an overwhelming majority of the time, a 10-incompressible string will be produced.

Hereafter, we will identify the random strings with sufficiently incompressible strings. Just how incompressible? We cannot give a precise answer, but for each c, for sufficiently long c-incompressible strings behave, for all practical purposes, like statistically random strings.

Given that it is easy to randomly produce an incompressible string, we might also ask whether we can produce incompressible strings algorithmically. Let $\mathscr{S}_c$ be the collection of c-incompressible strings. Is there a computable function $f : \mathbb{N} \to 2^{<\omega}$ such that $ran(f) \subseteq \mathscr{S}_c$?

Definition 3.3. *A set $S \subseteq 2^{<\omega}$ is* immune *if it is infinite and has no infinite computably enumerable subset.*

We will show the following, which immediately implies a negative answer to the above question.

Theorem 3.4. *For each $c \in \mathbb{N}$, the collection of c-incompressible strings is immune.*

Proof. Fix $c \in \mathbb{N}$. First note that $\mathscr{S}_c$ is infinite, since by Proposition 3.2, there is a c-incompressible string of every length.

Now suppose that $\mathscr{S}_c$ contains an infinite computably enumerable subset $\{\tau_0, \tau_1, \dots\}$. That is, there is some (total) computable function $f : \mathbb{N} \to 2^{<\omega}$ such that $f(i) = \tau_i$ for every i. Without loss of generality, we can assume that $|\tau_i| \leq |\tau_{i+1}|$ for every i. Indeed, we can define a function $\hat{f} : \mathbb{N} \to 2^{<\omega}$ that copies f as long as f enumerates a string with length longer than the lengths of the previously enumerated strings. That is, if f enumerates a string of length less than or equal to a previously enumerated string, $\hat{f}$ will not enumerate this string but will wait for a longer string to be enumerated by f. Since $ran(\hat{f}) \subseteq ran(f) \subseteq \mathscr{S}_c$, we can assume that f has this property (replacing f with $\hat{f}$ if it does not). Note that $|\tau_i| \leq |\tau_{i+1}|$ for every i implies that $|\tau_i| \geq i$ for every i.

Now, using the function f, we can define a Turing machine M_f that maps n (written in binary) to $f(n)$ for each n. Then

$$C_{M_f}(\tau_n) \leq \lfloor \log_2(n) \rfloor + 1$$

and hence

$$C(\tau_n) \leq \lfloor \log_2(n) \rfloor + d \tag{1}$$

for some $d \in \mathbb{N}$. However, since each τ_n is c-incompressible,

$$C(\tau_n) \geq |\tau_n| - c \geq n - c, \tag{2}$$

where the latter inequality follows from the observation at the end of the previous paragraph. But for sufficiently large n,

$$n - c > \lfloor \log_2(n) \rfloor + d,$$

which, together with (1) and (2), yields a contradiction. Thus, $\mathscr{S}_c$ contains no infinite computably enumerable subset. □

Corollary 3.5. *There is no total computable $f : \mathbb{N} \to 2^{<\omega}$ whose range consists only of c-incompressible strings.*

Corollary 3.6. *The function $C : 2^{<\omega} \to \mathbb{N}$ is not computable.*

Proof. If we could compute the value $C(\sigma)$ for each $\sigma \in 2^{<\omega}$, we could immediately determine which strings are c-incompressible (for any fixed c). Then we could define a computable enumeration of *all* c-incompressible strings, which is impossible. □

One other consequence of Theorem 3.4 is that we have a second example of a set that is computably enumerable but not computable (the first being the halting problem).

Theorem 3.7. *For each $c \in \mathbb{N}$, the collection of c-compressible strings is computably enumerable but not computable.*

Proof. Recall that a set $S \subseteq 2^{<\omega}$ is computable if and only if S and $2^{<\omega} - S$ are both computably enumerable. Since the collection of c-incompressible strings is immune, it cannot be computably enumerable. It follows that the collection of c-compressible strings is not computable.

To show that this collection is computably enumerable, observe that $\sigma \in 2^{<\omega}$ is c-compressible if and only if

$$(\exists \tau)(\exists s)(U(\tau)\!\downarrow \text{ in } < s \text{ steps } \;\&\; U(\tau) = \sigma \;\&\; |\tau| < |\sigma| - c).$$

Note that the relation $R(\sigma, \tau, s)$ given by

$$U(\tau)\!\downarrow \text{ in } < s \text{ steps } \;\&\; U(\tau) = \sigma \;\&\; |\tau| < |\sigma| - c$$

is a computable relation. Hence, by a lemma from the previous chapter, the collection of c-compressible strings is computably enumerable.

□

4. Kolmogorov complexity and incompleteness

In this final section, we will prove yet another incompleteness theorem known as Chaitin's Incompleteness Theorem (CIT), which provides us with infinitely many undecidable sentences in the language of arithmetic. We also will use CIT to provide a new proof of G2 by formalizing the paradox of the surprise exam (explained below).

In order to prove CIT, we need to express statements about the Kolmogorov complexity of strings in Peano arithmetic. First, we define a map $\# : 2^{<\omega} \to \mathbb{N}$, which sends each string to a unique natural number that will serve as its code. Let

$$\#(\sigma) = n \iff 1^\frown\sigma \text{ is the binary expansion of } n+1,$$

where $1^\frown\sigma$ is the string obtained by concatenating 1 and σ. Thus we have,

$$\begin{aligned} \#(\epsilon) &= 0 \\ \#(0) &= 1 \\ \#(1) &= 2 \\ \#(00) &= 3 \\ \#(01) &= 4 \\ \#(10) &= 5 \\ \#(11) &= 6 \\ &\vdots \end{aligned}$$

We also define a map $|\cdot| : \mathbb{N} \to \mathbb{N}$ such that if $n = \#(\sigma)$, then $|n|$ is the length of the string σ; that is, $|\#(\sigma)| = |\sigma|$.

Next, to represent Kolmogorov complexity in the language of arithmetic, note that since every partial computable function is definable in *PA*, there is some formula $\psi(x, y)$ (of the form $(\exists z)\theta(x, y, z)$, where θ has no unbounded quantifiers)) such that

$$U(\sigma) = \tau \iff PA \vdash \psi(\underline{\#(\sigma)}, \underline{\#(\tau)}).$$

Note that $C(\sigma) \leq n$ if and only if there is some τ such that $|\tau| = n$ and $U(\tau) = \sigma$. Thus, using our coding of members of $2^{<\omega}$ and the representations of $|\cdot|$ and U, we can express this by a formula in the language of arithmetic; for ease of presentation, we will simply write this formula as $C(\sigma) \leq n$. We can now state Chaitin's Theorem.

Theorem 4.1. *If PA is consistent, then there is some $L \in \mathbb{N}$ such that*

$$PA \nvdash C(\sigma) \geq L$$

for any $\sigma \in 2^{<\omega}$ and hence for all $n \geq L$,

$$PA \nvdash C(\sigma) \geq n$$

for any $\sigma \in 2^{<\omega}$.

We first prove a lemma.

Lemma 4.2. *If PA is consistent, then for $n \in \mathbb{N}$ and $\sigma \in 2^{<\omega}$,*

$$PA \vdash C(\sigma) \geq n \Rightarrow \mathbb{N} \models C(\sigma) \geq n.$$

That is, if PA proves that the complexity of σ is at least n, then it is really the case that the complexity of σ is at least n.

Proof. Suppose that $PA \vdash C(\sigma) \geq n$ but in fact we have that $\mathbb{N} \models C(\sigma) < n$. The sentence expressing that $C(\sigma) < n$ has the form $(\exists x)\theta(x)$. However, in the last chapter, we stated that for any such sentence that

$$\mathbb{N} \models (\exists x)\theta(x) \Rightarrow PA \vdash (\exists x)\theta(x).$$

Hence $PA \vdash C(\sigma) < n$, but this contradicts our assumption that *PA* is consistent. □

Proof of CIT. Suppose that for every $n \in \mathbb{N}$, there is some $\sigma \in 2^{<\omega}$ such that

$$PA \vdash C(\sigma) \geq n.$$

We define a machine $M : 2^{<\omega} \to 2^{<\omega}$ that on input τ enumerates theorems of *PA* until it finds a proof of $C(\sigma) \geq k$ for some σ and some $k > 2|\tau|$. Then M outputs σ.

If $M(\tau)\downarrow = \sigma$, then $PA \vdash C(\sigma) \geq k$ for some $k > 2|\tau|$, and hence by by Lemma 4.2, it follows that $C(\sigma) \geq k > 2|\tau|$. By the Invariance Theorem, there is some $d \in \mathbb{N}$ such that

$$C(\sigma) \leq C_M(\sigma) + d$$

for every $\sigma \in 2^{<\omega}$. Now, if we give M some input δ with $|\delta| = d$, we are guaranteed by our initial assumption to find some $\sigma \in 2^{<\omega}$ such that $C(\sigma) > 2|\delta| = 2d$. However, since $M(\delta)\downarrow = \sigma$ we also have

$$C(\sigma) \leq C_M(\sigma) + d \leq |\delta| + d = 2d,$$

which is impossible. Thus it must be the case that *PA* can only prove $C(\sigma) \geq n$ for finitely many n, bounded by some $L \in \mathbb{N}$. □

Note that whereas Gödel's first incompleteness theorem gives us one undecidable sentence, Chaitin's theorem gives us infinitely many. However, the Gödel sentence can be produced computably, whereas the undecidable sentences in Chaitin's theorem cannot be produced computably, as we cannot computably determine the bound L.

We now turn to a version of G2 that follows from CIT. Just as the proof of G1 can be formalized within arithmetic, the proof of CIT can be formalized within arithmetic, resulting in:

$$PA \vdash \neg Prov(\ulcorner \underline{0} = \underline{1} \urcorner) \to (\exists L)(\forall x)(|x| = L \to \neg Prov(\ulcorner C(x) \geq L \urcorner)).$$

As in the original proof of G2, we assume $PA \vdash \neg Prov(\ulcorner \underline{0} = \underline{1} \urcorner)$ and derive a contradiction. To do so, we formalize the surprise exam paradox.

Suppose that a teacher tells his/her class, “There will be a surprise exam next week, and you will be surprised when it comes." The students conclude that there will be no surprise exam on the basis of the following line of reasoning.

— If the exam were on Friday, then when class ends on Thursday and we have yet to take the exam, we would know it is on Friday and thus it would not be a surprise.

— If the exam were on Thursday, then when class ends on Wednesday and we have yet to take the exam, since it cannot be on Friday, we would know it is on Thursday and thus it would not be a surprise.

— Similarly, the exam cannot be on Wednesday nor Tuesday. But if the exam were on Monday, since it cannot be on any other day of the week, we would know that it is on Monday and thus it would not be a surprise.

— Therefore there is no surprise exam next week.

The following week, the teacher gives a surprise on Wednesday, and the students are all surprised.

How do we formalize this paradox? Let N be the number of strings σ of length L such that $C(\sigma) \geq L$, where L is the bound from CIT (so that PA cannot prove that any string has complexity $\geq L$). Note that $N \leq 2^L$. Moreover, we proved that there is one 0-incompressible string of each length, and thus $N \geq 1$. This argument can be formalized in PA, and hence $PA \vdash N \geq 1$.

Here we can think of 2^L as corresponding to the days of the week on which the surprise exam might be held, while N corresponds to the day of the week on which the exam is held. Just as the students reason in such a way that they cannot determine the day of the week on which the exam will be held, in PA, one can show that PA cannot determine which of the possible values $1, \dots, 2^L$ N is equal to.

That is, we prove inductively for each $i = 1, \dots, 2^N$ that if $PA \vdash N \geq i$, then $PA \vdash N \geq i+1$. We will not fully write out the formal details here (the full details can be found here). For the base case, we already have that $PA \vdash N \geq 1$. Suppose now that $PA \vdash N \geq i$.

(1) First, PA can prove that if $N = i$ then there are $2^L - i$ distinct strings σ with $C(\sigma) < L$.

(2) Since PA proves all true statements of the form $C(\sigma) < L$, it follows that PA can prove that if $N = i$ then it is provable that there are $2^L - i$ distinct strings σ with $C(\sigma) < L$.

(3) Moreover, PA can prove that if $N \geq i$, then if there are $2^L - i$ distinct strings σ with $C(\sigma) < L$, there is some string τ with $C(\tau) \geq L$.

(4) By the derivability conditions (discussed in the proof of G2), it follows that PA can prove that if $N \geq i$ is provable, then if it is provable that there are $2^L - i$ distinct strings σ with $C(\sigma) < L$, there it is provable that there is some string τ with $C(\tau) \geq L$.

(5) By (2) and (4) one can conclude (using some basic logic) that if PA can prove that if $N = i$ and it is provable that $N \geq i$, then it is provable that there is some string τ with $C(\tau) \geq L$.

(6) Since $PA \vdash N \geq i$ by the inductive hypothesis, it follows that PA proves that $N \geq i$ is provable.

(7) Hence by (5) and (6), PA proves that if $N = i$, then it is provable that there is some string τ with $C(\tau) \geq L$.

(8) Since it is not provable that there is some string τ with $C(\tau) \geq L$ (by proving CIT within arithmetic, as discussed above), it follows that $PA \nvdash N = i$, and thus $PA \vdash N \geq i+1$.

It now follows by induction that $PA \vdash N \geq 2^L + 1$, but since $N \leq 2^L$, we also have $PA \vdash N \leq 2^L$, which is impossible. Thus PA cannot prove its own consistency.