Appendix A

A Rapid Introduction to Mathematical Logic

> *"...our interest in formalized languages being less often in their actual and practical use as languages than in the general theory of such use and its possibilities in principle."* [Chu56, page 47]
>
> *"The formal systems of logic were created in order to be studied, not in order to be used. It is an interesting exercise to try to formalize Hardy and Wright's number theory book in Peano Arithmetic. Any logician will see that it can theoretically be done. But to do it in practice is far too cumbersome... This has not bothered logicians, who...have not been interested in actually formalizing anything, but only in the possibilities of doing so."* [Bee86, pages 53–54]

Formal methods are grounded in mathematical logic, and some familiarity with the main concepts and terminology of that field are essential to an understanding of formal methods. Unfortunately, most textbooks on formal methods introduce only the fragments of logic relevant to their particular methods and leave the reader uninformed of alternatives. Textbooks on mathematical logic are also unsatisfactory for those seeking an introduction to the background of formal methods: elementary texts such as [Hod77, Cop67, Lem87], while excellent for their intended purpose, omit much that is of importance concerning logic as a foundation for formal methods in computer science, while more advanced texts such as [Chu56, End72, Kle52, Men64, Mon76, Sho67] are rather challenging, yet still omit some of the topics necessary for an appreciation of the mechanization of mathematical logic in support of formal methods (while including many that are unnecessary for that purpose).

To help readers gain the necessary background, I provide in this appendix a minimal introduction to mathematical logic without all the technicalities usually

introduced in logic texts. My main goal is simply to acquaint the reader with the main concepts and terminology of the field. There is, however, one important topic where the going does get a little technical: this is the explanation of *interpretation* and *model* in Sections A.2, A.3, and A.4. An appreciation of these concepts is necessary in order to understand the relationship between "true" and "provable," and between the notions "sound" and "complete," and I urge readers to whom this material is new to persevere with it. Recent controversies, such as that surrounding the British Viper microprocessor (see Section 2.5.2 in the main text) show that the formal notion of "proof" is no longer of merely specialist interest: all those concerned with formal methods and certification should understand the technical meaning of "proof" as used in logic. Later sections that deal with somewhat esoteric topics that can be skipped at first reading are marked with the "dangerous bend" sign  .

A secondary goal of this presentation is to sketch the origins and motivations for some of the foundational approaches taken in mathematical logic, and to identify some of the alternatives that are often omitted from elementary introductions to logic. The reason for doing this can be found in the quotations at the head of this chapter: mathematical logic was developed by mathematicians to address matters of concern to them. Initially, those concerns were to provide a minimal and self-evident foundation for mathematics; later, technical questions about logic itself became important. For these reasons, much of mathematical logic is set up for *meta*mathematical purposes: to show that certain elementary concepts allow some parts of mathematics to be formalized *in principle*, and to support (relatively) simple proofs of properties such as soundness and completeness. Formal methods in computer science, on the other hand, is concerned with formalizing requirements, designs, algorithms and programs, and with developing formal proofs *in practice*. Thus, formal systems and approaches that are very suitable for mathematicians' purposes may not be so suitable as a foundation for formal methods in computer science—especially when mechanized checking of formalizations is desired. This is particularly the case, in my opinion, with axiomatic set theory[1]—yet many books on formal methods take their lead from the mathematicians and use axiomatic set theory and Hilbert-style formalizations of predicate calculus as a foundation. Those who are curious about alternatives and might want to learn something about type theory, or the sequent calculus, say, must generally turn to rather advanced texts on mathematical logic. So although this Appendix cannot provide an adequate introduction to these alternatives, it does at least sketch them, and describe the motivations behind the various different approaches, and some of their strengths and weaknesses. I hope that readers who persevere to the end will find this mate-

[1]This is not to say that the notation of set theory is not useful in formal methods; the question is whether set theory should be the *foundation* for formal methods.

rial helps them to become more knowledgeable and critical in their use of formal methods.

A.1 Introduction

Logic is concerned with methods for sound reasoning. More particularly, *mathematical* or *formal* logic is concerned with methods that are sound because of their *form*, and independent of their content. For example, the deduction

> That jaberwocky is a tove;
> All toves are slithy;
> Therefore that jaberwocky is slithy

seems perfectly sound, because of its form, even though we have no idea what a *jaberwocky* or a *tove* might be, nor what it means to be *slithy*. On the other hand, the deduction

> That plane is a Boeing 737;
> Therefore it has two engines

is not logically sound, even though its conclusion is true, because the line of reasoning employed is not generally valid—it jumps to a conclusion that is not supported by the facts *explicitly* enumerated. The argument used has the same form as

> That car is a Chrysler;
> Therefore it has two engines,

which is palpable nonsense. We can correct the problem by adding a premise to make explicit the "knowledge" used in coming to the conclusion:

> That plane is a Boeing 737;
> All Boeing planes, except the 747, have two engines;
> Therefore that plane has two engines.

The line of reasoning is now sound; it doesn't depend on the meaning of "Boeing 737" and the other terms appearing in its statement. To see this, we can abstract the argument to the form

> That A is an X;
> P is true of all A's, except Y;
> Therefore P is true of that A

and observe that it seems to be sound independently of what A, X, Y, or P mean.[2]

Note, however, that although the soundness of this line of reasoning does not depend on what a "Boeing 737" is, it depends crucially on what "all," "except," and "and" mean. Logic chooses some elements of language (such as "all," and "and"), gives them a very precise meaning, and then investigates the lines of reasoning and argument that can be built up from those elements; in particular, logic seeks to identify those lines of reasoning that are valid independently of the meaning of the other (so-called proper) terms that appear in the argument. Logics differ from one another in their choice of primitive elements, in the ways in which they allow terms to be combined, and in the lines of reasoning that they permit.

Notice that a valid or sound argument does not guarantee that the conclusion we draw will be true. For example, if the plane we saw was a Boeing 707 (or 727), the perfectly valid pattern of deduction given above would cause us to deduce, incorrectly, that the plane has only two engines. A sound method of reasoning guarantees true conclusions only if the premises are true statements. The truth of a statement such as

> All Boeing planes have two engines, except the 747

depends on the real-world interpretation we place on the terms such as "Boeing planes." If the interpretation we choose is "all Boeing passenger planes in current service," then this premise to our argument is false, and a perfectly valid line of reasoning can lead us to a false conclusion.

Two sources of error can lead us to draw false conclusions about the world: we can reason correctly from incorrect premises, and we can reason incorrectly from correct premises (and, of course, we can also go doubly wrong and reason incorrectly from bad premises). The contribution of mathematical logic is that it gives us the tools to eliminate the second of these ways to draw incorrect conclusions. Logic tells us what follows from what, so that we can be sure that our conclusions will be at least as good as our premises. However, no formal method can assure us that our premises are true statements in the intended interpretation; we have to introspect, discuss them with others, test their consequences, and generally use informal methods to accomplish this important task.[3]

[2] Though it does depend on knowing that an X is not a Y (i.e., a 737 is not a 747). When we come to formalize this (in Section A.4), we will have to make even this "obvious" fact explicit.

[3] However, formal methods provide a precise and standardized notation, a repertoire of standard constructions, and well-accepted sets of premises (i.e., axiomatizations) for many familiar notions. In addition, formal methods can help us establish that our premises are mutually consistent, and can support systematic exploration of their consequences (through attempts to prove properties that ought to be true if the premises are correct). Thus the informal process of validating premises can be made more systematic using formal methods.

The practical value of mathematical logic arises when we use long or difficult lines of reasoning to draw an important conclusion. If we can satisfy ourselves, by informal methods, that the premises to our argument are true in our intended interpretation, then mathematical logic can assure us that the conclusion will also be true in that interpretation.

> "The true method should provide us with...a certain sensible material means which will lead the intellect, like the lines drawn in geometry and the forms of operations which are prescribed to novices in arithmetic. Without this, our minds will not be able to travel a long path without going astray." Leibniz, quoted in [Bou68, page 303].

Mathematical logic—Leibniz' "true method"—provides us with rules of deduction, analogous to those of arithmetic, that allow the validity of an argument to be checked by a process akin to calculation.

A.2 General Concepts

In this section I give more detailed and precise explanations to some of the notions introduced above.[4] The explanations are given in terms of the abstract notion of "formal system" and are generic to all logics; I give examples of specific logics and theories in the subsections that follow.

The first thing that must be specified about a particular formal system is the language in which its statements are expressed. Just as with a programming language, there must be some grammar that defines the syntactic form of the things we can write in the system. Mathematicians tend to prefer very terse, succinct languages, so the statements in most of their formal systems look very cryptic to the untrained eye. Legal strings of symbols in the language of a formal system are called *well-formed formulas* or *wff*s. Wffs that are "self contained" (technically, contain no free variables) are called sentences.

Formal systems come in families, and there are usually two parts to a formal system: one part that is common to all the systems in a given family, and one part that changes from one member of the family to another. The first part is called the *logical* part of the system, and the second the *nonlogical* or *extralogical* or *proper* part. (I will use only the first of these alternative terms). For example, all *first-order theories* have a common logical part variously called *first-order logic* or *first-order predicate calculus*, or *elementary logic* or *quantification theory*. (I will use the simple name *predicate calculus*.)

[4] My treatment draws on [Tar76, Dav89].

Those symbols of the system that belong to the logical part are called the *logical* symbols of the system. The other symbols are called the *nonlogical* symbols of the formal system. The logical symbols generally include the *propositional connectives* "not" (a monadic operator, usually written as prefix $\neg$ or $\sim$ by mathematicians), "and" (a dyadic operator, often called *conjunction* and written as infix $\wedge$), "or" (dyadic, often called *disjunction* and written as infix $\vee$), "implies" (dyadic, written as infix $\supset$, $\rightarrow$, or $\Rightarrow$), and "if and only if" (dyadic, sometimes called *equivalence*, and written as infix *iff*, $\leftrightarrow$, $\Leftrightarrow$, or $\equiv$). More powerful logics include *function* symbols (such as f, g, and some that may be given a built-in interpretation such as $+$), predicate symbols (such as P, Q, and some that may be given a built-in interpretation such as $=$ and $<$), the *quantifiers* "for all" ($\forall$) and "there exists" ($\exists$), and/or *modalities* such as "necessarily" ($\Box$) and "possibly" ($\Diamond$).

The propositional connectives developed out of attempts to understand and codify principles of sound argument, and were intended to capture important constructions used in natural language. Prior to the development of modern formal logic, the usages of natural language were arbiters of what the connectives should mean, and there could be disagreement over this: for example, some medieval logicians treated "or" as inclusive ("either one or the other or both"), while others regarded it as exclusive ("either one or the other, but not both"). In formal logic, the meanings of the operators are determined by the rules of deduction (as explained shortly) and do not depend on intuitive notions.

Although the meanings of the logical operators do not *depend* on intuitive notions, they should clearly *correspond* to them—since logic is intended to support and strengthen our innate capabilities, not oppose them. Most of the logical operators do directly correspond to familiar notions, but the "implies" operator is a little less straightforward.

The symbol $\supset$ of logic is sometimes called *material* implication in order to distinguish it from the unadorned term "implication" used in ordinary discourse. The intent is that $A \supset B$ should capture conditional constructions of the form "*if* A *then* B." From this intuition, it follows that if A is a true statement, then $A \supset B$ should be considered true only if B is also a true statement. Thus

$$2 + 2 = 4 \supset \text{Paris is the Capital of France}$$

is a true sentence (because both the *antecedent* or *condition* "$2 + 2 = 4$" and *consequent* or *conclusion* "Paris is the Capital of France" are true). Observe that this example indicates one of the unintuitive aspects of material implication: there need be no "causal" connection between the antecedent and consequent. In natural language, we generally expect

some connection or association between the clauses of a conditional, such as "*if* Paris is the seat of French Government *then* Paris is the Capital of France."

Leaving these doubts aside, and proceeding with our examination of material implication, we can see that

$$2 + 2 = 4 \supset \text{London is the Capital of France}$$

is a false sentence (because the antecedent is true but the consequent is false). But what if A, the antecedent is false? Should

$$2 + 2 = 5 \supset \text{Paris is the Capital of France}$$

be considered a true sentence? And how about

$$2 + 2 = 5 \supset \text{London is the Capital of France?}$$

In natural language one could make a plausible case for any of the alternatives (or even argue that a conditional need be neither true nor false when its antecedent is false), but in logic we adopt the convention that if A is false, then $A \supset B$ is true, whatever the value of B. One way to see that this is reasonable is to consider the alternatives. We know that when A is true, $A \supset B$ must be true exactly when B is true. To complete the definition, we need to assign truth values to $A \supset B$ when A is false, and there are just four possible ways to do this: (1) it could be false whatever the value of B, (2) it could be true exactly when B is true, (3) it could be true exactly when B is false, and (4) it could be true whatever the value of B. If we choose the first of these alternatives, then $A \supset B$ is exactly the same as $A \wedge B$ (i.e., "implies" is the same as "and"); if we choose the second, then $A \supset B$ is the same as B and the value of A is irrelevant; and if we choose the third, then $A \supset B$ turns out to be the same as $A \equiv B$. None of these corresponds to any reasonable interpretation of "implies" or "*if*...*then* ...," and so we are left with the fourth alternative.

Although this treatment of material implication works perfectly well, some logicians remain unhappy that it does not correspond exactly to informal usages and have sought different formulations, such as the notions of "strict" and "relevant" implication. The consideration of alternatives that we conducted above shows that any satisfactory formulation that is different to material implication will require modification of the basic logic, not just tinkering with the operators. Consequently, strict implication requires modal logic (which is described later), and relevant implication requires relevance logic [Dun85]. Material implication may

seem a little strange at first, but it soon becomes familiar.[5] Alternative formalizations of conditional sentences are of philosophical and technical interest, but do not rival the practical utility and importance of the classical formulation in terms of material implication.

In addition to its language, a formal system specifies a selected (possibly infinite) set of sentences that are taken as given and called *axioms*, and a set of *rules of inference*, which are ways of deducing new sentences from given sets of sentences. Axioms are divided into logical and nonlogical axioms; rules of inference usually belong to the logical part of the system. The axioms and the rules of inference constitute the *deductive system* of the formal system considered.

Axioms are generally specified by *axiom schemes* (mathematicians often say *schema* and *schemata*), such as

$$\phi \vee \neg\phi,$$

where ϕ stands for an arbitrary sentence. This example (it is the *law of the excluded middle*) states that for any sentence ϕ, either ϕ or its negation must be true (there is no middle ground). When we put a particular sentence (e.g., "it will rain tomorrow") in place of ϕ, we obtain a specific axiom as an *instance* of the axiom scheme, such as

$$\text{it will rain tomorrow} \vee \neg \text{ it will rain tomorrow.}$$

Axiom schemes are convenient, because they allow an infinite number of axioms to be specified in a finite manner.

Similarly, rules of inference are often written in the form of schemes, exemplified by

$$\frac{\phi,\ \phi \supset \psi}{\psi},$$

where ϕ and ψ stand for arbitrary sentences. This example (it is the rule known as *modus ponens*) states that from a pair of sentences, one of the form ϕ and one of the form $\phi \supset \psi$, we may deduce the sentence ψ. We say that ϕ and $\phi \supset \psi$ are the *premises* or *antecedents* to this rule of inference, and ψ is its *conclusion* or *consequent*.

Notice that there are two mathematical languages in use here. One, the *object language*, is the formal system we are trying to define; the other is the *metalanguage* that we are using to do the definition. Here, the object language is a fragment of propositional calculus, and our metalanguage is the traditional language of informal

[5] Particularly once some important laws are learned, such as that $A \supset B \supset C$ is parsed as $A \supset (B \supset C)$, and is equivalent to $(A \wedge B) \supset C$. Some of these laws are listed in Section A.3.

mathematics expressed in English. It is necessary to keep the distinction between object language and metalanguage clearly in mind, especially when the same symbols are used in both; logicians use the terms *use* and *mention* to distinguish these cases. When we talk *about* some object, we are said to mention it; when we use an object to *refer* to some other object, we are said to use it. No confusion is likely with physical objects (we do not use the planet Mars to refer to something else), but it is possible with names or symbols. For example, from

"Toby is a short haired cat" and
"Toby is a name with four letters"

we might conclude that a certain name with four letters is a cat [Cur77]. In order to avoid this absurdity, we must recognize that "Toby" is used in quite different senses in these two sentences: in the first it is *used* to refer a certain cat, while in the second it is *mentioned* as a particular word. In order to reduce confusion, symbols are often enclosed in quotes when they are mentioned rather than used:

"'Toby' is a name with four letters"

The variables ϕ and ψ used in the examples given earlier are *metalinguistic* variables, since they stand for arbitrary sentences of the object language. Axiom schemes and the operation of substituting sentences for metavariables are elements of the metalanguage; the resulting axioms are part of the object language. Similarly, it is necessary to distinguish proofs and theorems *about* the object language (i.e., metaproofs and metatheorems) from proofs and theorems *in* the object language. I am now ready to define the latter kind of proofs and theorems.

A *proof* of a sentence ψ from a set of sentences Γ is a finite sequence $\phi_1, \ldots, \phi_n$ of sentences with $\phi_n = \psi$, and in which each ϕ_i is either an axiom, or a member of Γ, or else follows from earlier ϕ_j's by one of the rules of inference. We say that ψ is provable from the *assumptions* Γ (or simply "Γ proves ψ") and write $\Gamma \vdash \psi$.[6] A *theorem* is a sentence that is provable without assumptions, and we write simply $\vdash \psi$.[7] The *theory* of a given formal system is the set of all its theorems.

A system is *consistent* if it contains no sentence ϕ such that both ϕ and its negation are theorems (i.e., if its theory does not contain both ϕ and $\neg\phi$). It is a metatheorem of all the logics considered here that *all* sentences are provable in an inconsistent system, and such systems are therefore useless.

[6] The symbol $\vdash$ is usually pronounced "turnstile."

[7] The axioms can be thought of as assumptions that come "for free." The reason we add (nonlogical) axioms to a formal system, rather than simply carry them as assumptions to all our theorems, is that there is generally an infinite number of axioms, and we need the metalinguistic notion of scheme to represent them conveniently. Of course, we could introduce a notion of "assumption scheme," but this would not be the way things have traditionally been done.

A system is *decidable* if there is an algorithm (i.e., a computer program that is guaranteed to terminate) that can determine whether or not any given sentence is a theorem. A system is semidecidable if there is an algorithm that can recognize the theorems (i.e., the algorithm is guaranteed to halt with the answer "yes" if given a theorem, but that need not halt if given a nontheorem—though if it does halt, it must give the answer "no" in this case). A system is *undecidable* if it is neither *decidable* nor *semidecidable*.

Notice that "proof" and "theorem" are purely syntactic notions: that is, they are marks on paper (or symbols in a computer). The formal system provides rules for transforming one set of marks into another; a proof is simply a sequence of formulas that conforms to the rules and ends up with a series of marks that we call the theorem. We do not have to know what any of these marks "mean" in order to tell whether or not a purported proof really is a correct proof of a given theorem; we simply have to check whether it follows the rules of the formal system concerned. Of course, the rules of formal systems are not chosen arbitrarily; they are chosen in the hope that if we interpret the marks on paper in some consistent way as statements about the real world, then the theorems of the system will be *true* statements about the world. The relationship between language and the world is the concern of *semantics*, and the goal of mathematical logic is to set things up so that the syntactic notion of *theorem* coincides with the semantic notion of *truth*. If this can be done, and if our axioms and assumptions correspond to true statements about (some aspect of) the real world, then the theorems that we can prove will also correspond to true statements about the world. In other words, we will be able to deduce (new) true facts about the world from given true facts simply by following the syntactic rules of our chosen formal system. We do not need to "understand" the world in order to carry out these syntactic operations; all our "understanding" (of the particular domain of interest) is encoded in the nonlogical axioms of the formal system used (and in the assumptions, if any, of the particular proofs performed). Since no "understanding" of the world is required to construct (or check) the purely syntactic objects that are proofs, these operations can, in principle, be performed by machine. Understanding of the world is needed only to choose the nonlogical axioms (and assumptions) and to interpret the utility of the theorems proved. Thus, the attraction of formal methods is that the necessary intuition and understanding of the world (i.e., the properties of the application domain concerned) are recorded *explicitly* in axioms and assumptions that can be subjected to intense scrutiny, and all the rest follows by deductions that can be mechanically checked.

The connection between syntax and semantics is established by an *interpretation* that identifies some real-world "domain" $\mathcal{D}$ and associates a true or false statement about $\mathcal{D}$ with each sentence. The statement associated with sentence ϕ is called its *valuation* in $\mathcal{D}$. Interpretations must satisfy certain structural properties that depend on the formal system concerned. (For example, if the formal system has

function symbols, then the interpretation of $f(x, y)$ must be determined by the interpretations of f, x, and y.)

An interpretation is a *model* for a formal system if it valuates all its axioms to true; in addition, an interpretation is a model for a set of sentences T if, as well as being a model for the formal system concerned, it also valuates all the sentences in T to true. A set of sentences is *satisfiable* if it has a model. We say that a set of sentences T *entails* a sentence ψ and write $T \models \psi$ if every model of T is also a model of ψ—that is, if ψ valuates to true in every model of T. We say a sentence ψ is (universally) *valid* and write $\models \psi$ if it valuates to true in all models of its formal system. This connection between syntax and semantics, and the corresponding relationship between "provable" and "valid" were known and used informally (e.g., by Gödel and Skolem) in the 1920s; they were formalized by Tarski in the early 1930s.

A formal system is *sound* if $\Gamma \models \psi$ whenever $\Gamma \vdash \psi$; it is *complete* if $\Gamma \vdash \psi$ whenever $\Gamma \models \psi$. Roughly speaking, soundness ensures every provable fact is true, completeness ensures every true fact is provable. Notice that an inconsistent system cannot be sound. Obviously, only sound systems are of use in formal methods (or mathematics).[8] It would be nice if they were also complete (and even nicer if they were decidable), but I will observe later that most interesting formal systems are incomplete (and very few are decidable).

In some formal systems, additional constraints are placed on the selection of models, and it is sometimes necessary to distinguish between soundness and completeness results relative to *all* models and those relative to certain *preferred* models. Other formal systems are constructed in the hope that they capture some aspect of reality exactly—that is, in the hope that they will have only one model, the *intended* or *standard* one. For example, the postulates of Euclid were intended to capture "ordinary" geometry exactly. Formal systems that have only one interpretation (up to isomorphism) are called *categorical*. In general, it is not possible to limit theories to just a single model; most interesting theories have *unintended* or *nonstandard* models in addition to the intended or standard ones (see section A.5.2).

Having established the general framework, I will now consider some specific formal systems that are of particular importance.

[8] It was a goal of the formalist school, led by Hilbert, to demonstrate (using only elementary methods) that the formal systems proposed as foundations for mathematics (such as set theory with the Axiom of Choice—see Section A.5.4) are sound. Gödel's results (see Section A.5.2) destroyed this plan. Soundness of the formal systems underlying mathematics cannot be demonstrated conclusively within those same systems, and must ultimately appeal to our experience and intuition.

A.3 Propositional Calculus

The simplest formal system generally considered is the *propositional calculus*, whose purpose is to provide a meaning for the propositional connectives listed earlier (Subsection A.2).[9] In its leanest form, the nonlogical part of the propositional calculus is empty, and its logical part considers only the connectives "implies" (written here as infix $\supset$) and "not" (written here as prefix $\neg$). The rest of its language consists of the parentheses, and some set of *proposition symbols*, which mathematicians generally write as P, Q, R, but which can be arbitrary strings such as "the cat is on the mat" that are considered atomic (i.e., without internal structure). The grammar of propositional calculus defines its sentences as follows:

- Any proposition symbol is a sentence, and
- If ϕ and ψ are sentences, then so are $(\neg\phi)$ and $(\phi \supset \psi)$.

(This is the mathematicians' description of propositional calculus, in which every compound sentence is fully parenthesized. Mathematicians drop parentheses—and so will I—when they are deemed unnecessary according to some informal rules. Computer scientists might give a context-free grammar for the language and make all these details explicit.)

There are three axiom schemes for propositional calculus:

A1. $\phi \supset (\psi \supset \phi)$,

A2. $(\phi \supset (\psi \supset \rho)) \supset ((\phi \supset \psi) \supset (\phi \supset \rho))$,

A3. $(\neg(\neg\phi)) \supset \phi$[10]

and one rule of inference

$$\frac{\phi,\ \phi \supset \psi}{\psi}$$

(modus ponens, often abbreviated MP).

The axioms of propositional calculus are infinite in number, since there are instances of each of the three schemes for all possible combinations of propositions. The symbols ϕ, ψ, and ρ appearing in the axiom schemes (and rule of inference) above are metalinguistic variables (or *metavariables*). An axiom *instance* is obtained from one of the axiom schemes by replacing its metalinguistic variables by propositions in a consistent fashion. I indicate the axiom instance of scheme 1 in which P replaces ϕ and Q replaces ψ by writing $A1[\phi \leftarrow P, \psi \leftarrow Q]$.

[9] Sources used for this section include Barwise [Bar78b] and De Long [DeL70].

[10] Sometimes $((\neg\phi) \supset (\neg\psi)) \supset (\psi \supset \phi)$ is used instead.

Using this machinery, we can prove the theorem $\vdash (P \supset P)$.

1. $((P \supset ((P \supset P) \supset P)) \supset ((P \supset (P \supset P)) \supset (P \supset P)))$
   $A2[\phi \leftarrow P, \psi \leftarrow (P \supset P), \rho \leftarrow P]$
2. $(P \supset ((P \supset P) \supset P))$ $A1[\phi \leftarrow P, \psi \leftarrow (P \supset P)]$
3. $((P \supset (P \supset P)) \supset (P \supset P))$ MP on 1 and 2
4. $(P \supset (P \supset P))$ $A1[\phi \leftarrow P, \psi \leftarrow P]$
5. $(P \supset P)$ MP on 3 and 4.

This derivation is completely unintuitive; the advantage of the "Hilbert-style" axiomatization I have been using is that it allows the interpretations of the propositional calculus to be defined in a concise manner. This is done in the next couple of pages. Following that, I introduce an alternative system that allows proofs in propositional calculus to be developed in an intuitive manner.

The domain of the interpretations of the propositional calculus is the set of truth values $\{\mathcal{T}, \mathcal{F}\}$ ($\mathcal{T}$ is an abbreviation for truth, $\mathcal{F}$ for falsehood). An interpretation of the propositional calculus assigns exactly one of these truth values (called its *valuation*) to every proposition symbol. Let $v(\phi)$ denote the valuation given to the proposition ϕ in a particular interpretation. Then we require that the valuation given to compound propositions in that interpretation is derived by recursive application of the following two *truth tables*:

ϕ	$\neg\phi$
$\mathcal{T}$	$\mathcal{F}$
$\mathcal{F}$	$\mathcal{T}$

ϕ	ψ	$\phi \supset \psi$
$\mathcal{T}$	$\mathcal{T}$	$\mathcal{T}$
$\mathcal{T}$	$\mathcal{F}$	$\mathcal{F}$
$\mathcal{F}$	$\mathcal{T}$	$\mathcal{T}$
$\mathcal{F}$	$\mathcal{F}$	$\mathcal{T}$.

Sentences of the propositional calculus that valuate to $\mathcal{T}$ in all interpretations (i.e., the universally valid sentences) are called *tautologies*.

Interpretations constructed as described above provide models for the propositional calculus and it is possible to prove that the theorems and the tautologies coincide. Thus, the propositional calculus is sound (all theorems are tautologies) and complete (all tautologies are theorems). Furthermore, the propositional calculus is decidable—simply transfer into the semantic domain and use truth tables. Thus, another way to demonstrate $\vdash (P \supset P)$ is to construct the truth table for $(P \supset P)$ by substituting P for both ϕ and ψ in the general truth table for $\supset$ given above (note the second and third lines of the general table disappear, since they are impossible when ϕ and ψ have to take the same value)

P	$P \supset P$
$\mathcal{T}$	$\mathcal{T}$
$\mathcal{F}$	$\mathcal{T}$

and observe that its value is always $\mathcal{T}$. Hence, $\models (P \supset P)$, and so, by the completeness of propositional calculus, $\vdash (P \supset P)$.

The version of the propositional calculus I have given so far is rather impoverished. It is easy to add new connectives, either by treating them as abbreviations (i.e., macros), or by extending the formal system. Using macros, we would, for example, regard disjunction $(\phi \vee \psi)$ as simply an abbreviation for $(\neg\phi) \supset \psi$ and would replace all instances of $\phi \vee \psi$ by their corresponding expansions in both the syntactic and semantic domains. Observe that this is a metalinguistic approach.

Using the alternative approach of expanding the formal system, we would add $\vee$ to the language as a logical symbol, and add the corresponding rules of inference

$$\frac{\phi \vee \psi}{(\neg\phi) \supset \psi} \qquad\qquad \frac{(\neg\phi) \supset \psi}{\phi \vee \psi}.$$

Similarly, the interpretation of $\vee$ would be supplied by the truth table

ϕ	ψ	$\phi \vee \psi$
$\mathcal{T}$	$\mathcal{T}$	$\mathcal{T}$
$\mathcal{T}$	$\mathcal{F}$	$\mathcal{T}$
$\mathcal{F}$	$\mathcal{T}$	$\mathcal{T}$
$\mathcal{F}$	$\mathcal{F}$	$\mathcal{F}$.

It is usually a matter of taste whether the properties of new connectives are added to the formal system by a rule of inference or by an axiom scheme. For example, if the equivalence $(\equiv)$ connective has already been introduced, we can define $\vee$ by the axiom scheme

$$(\phi \vee \psi) \equiv ((\neg\phi) \supset \psi)$$

instead of the rule of inference given earlier. (Usually things are done in the opposite order, and $\phi \equiv \psi$ is viewed as an abbreviation for $(\phi \supset \psi) \wedge (\psi \supset \phi)$.)

There are a number of tautologies (often called "laws") that are worth learning by heart for use in systems that contain a generous collection of connectives:

		$\phi \vee \neg\phi$	law of the excluded middle
		$\neg(\phi \wedge \neg\phi)$	law of contradiction
$(\phi \vee \phi)$	$\equiv$	ϕ	laws of
$(\phi \wedge \phi)$	$\equiv$	ϕ	tautology
$\neg\neg\phi$	$\equiv$	ϕ	law of double negation
$\phi \supset \psi$	$\equiv$	$\neg\phi \vee \psi$	*verum sequitur ad quodlibet*
$\phi \supset \psi$	$\equiv$	$\neg\psi \supset \neg\phi$	law of contraposition
$\neg(\phi \vee \psi)$	$\equiv$	$\neg\phi \wedge \neg\psi$	De Morgan's
$\neg(\phi \wedge \psi)$	$\equiv$	$\neg\phi \vee \neg\psi$	laws
$\phi \vee (\psi \vee \mu)$	$\equiv$	$(\phi \vee \psi) \vee \mu$	Associative
$\phi \wedge (\psi \wedge \mu)$	$\equiv$	$(\phi \wedge \psi) \wedge \mu$	laws
$\phi \vee (\psi \wedge \mu)$	$\equiv$	$(\phi \vee \mu) \wedge (\phi \vee \mu)$	Distributive
$\phi \wedge (\psi \vee \mu)$	$\equiv$	$(\phi \wedge \mu) \vee (\phi \wedge \mu)$	laws
$\phi \supset (\psi \supset \mu)$	$\equiv$	$(\phi \wedge \psi) \supset \mu$	law of exportation
$(\phi \wedge \psi) \supset \mu$	$\equiv$	$(\phi \supset \mu) \vee (\psi \supset \mu)$	unnamed
$(\phi \vee \psi) \supset \mu$	$\equiv$	$(\phi \supset \mu) \wedge (\psi \supset \mu)$	laws

Notice that most of these laws come in pairs in which one member can be obtained from the other by interchanging $\wedge$ and $\vee$. Such pairs are called *duals*.

It is also worth remembering the derived inference rule

$$\frac{\neg\psi,\ \phi \supset \psi}{\neg\phi}$$

known as *modus tollens*. It follows from modus ponens by the law of contraposition.

Conventionally, the equivalence $\phi \equiv \psi$ is regarded as an abbreviation for a pair of "simpler" expressions, such as $(\phi \supset \psi) \wedge (\psi \supset \phi)$ or $(\neg\phi \wedge \neg\psi) \vee (\phi \wedge \psi)$ and it is often expanded into one of these forms whenever it is encountered during a proof. However, equivalence satisfies some beautiful laws, such as the *Golden Rule*:

$$(\phi \wedge \psi) \equiv \phi \equiv \psi \equiv (\phi \vee \psi)$$

(one of whose beautiful properties is that it does not matter how we associate—i.e., parenthesize—the equivalences). The book by Dijkstra and Scholten [DS90] make extensive and effective use of laws for equivalence.

The propositional connectives introduced so far are not the only ones. Those called **nand** and **nor** are defined by

$$\begin{aligned} \phi \ \textbf{nand}\ \psi &\equiv \neg(\phi \wedge \psi) \\ \phi \ \textbf{nor}\ \psi &\equiv \neg(\phi \vee \psi) \end{aligned}$$

and are often used in digital circuits. All the other propositional connectives can be defined in terms of either one of these two—a property first discovered by Pierce but often attributed to Sheffer (who wrote **nor** in the form $\phi \mid \psi$, where $\mid$ is known as "Sheffer's stroke").

Computer scientists generally like to have the three-way *if... then... else* connective available. It can be defined by

$$\textbf{if } \phi \textbf{ then } \psi \textbf{ else } \mu \equiv (\phi \supset \psi) \wedge (\neg\phi \supset \mu).^{11}$$

Finally, note that the constants *true* and *false* can be introduced by

$$\mathit{true} \equiv (\phi \vee \neg\phi),$$

$$\mathit{false} \equiv (\phi \wedge \neg\phi).$$

A.3.1 Propositional Sequent Calculus

The axiom schemes and rule of inference I have given for propositional calculus make for rather tedious proofs and are "barbarously unintuitive" [Hod83]. Usually, it is simpler to exploit the soundness and completeness of the propositional calculus and transfer into the semantic domain where theorems can be established using truth tables.[12] A case against doing this is that a proof should represent an *argument* for the truth a theorem and should lead to improved understanding (not least when the putative theorem is actually false and the proof attempt fails), whereas the truth-table method merely says "true" or "false." On the other hand, the propositional calculus is so limited, and its theorems so straightforward, that the opportunity

[11] In richer logics, *if... then... else* is often extended to a *polymorphic* form that denotes a value of a different sort (or type—these notions are explained in Sections A.9 and A.10) than a proposition. For example, in

$$|x| = \textbf{if } x < 0 \textbf{ then } -x \textbf{ else } x$$

the *if... then... else* denotes a numerical value. These polymorphic forms can be converted to the propositional kind by "lifting" the *if... then... else* over adjacent relations:

$$\textbf{if } x < 0 \textbf{ then } |x| = -x \textbf{ else } |x| = x.$$

[12] Large propositional expressions can arise in the specification and analysis of hardware circuits. Methods based on Boolean (or Binary) Decision Diagrams (BDDs) [Ake78, Bry86, Bry92] are generally the most efficient in practice by a considerable margin, and should be a basic component of any tool for formal methods that is intended to address such problems. Although BDDs are usually superior, methods derived from those of Davis and Putnam [DP60] can outperform BDDs on some problems [US93].

to gain insight from proofs is distinctly limited.[13] In richer systems, however, the ability to construct reasonably natural proofs in a reasonably effective manner will clearly be desirable, so I use the propositional calculus to illustrate one approach to doing this.

First, note that proofs can be simplified by using previously-proved theorems (which effectively enriches the set of sentences that can be used as axioms), and by deriving additional rules of inference. For example, there is an important metatheorem about propositional calculus called the *deduction theorem*, which I state as follows:

$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash (\phi \supset \psi)}$$

where Γ is any set of sentences and Γ, ϕ is interpreted as the union $\Gamma \cup \{\phi\}$. Notice that I have stated the Deduction Theorem as an extended rule of inference: formerly rules of inference were relations on sets of sentences; now I allow them to be relations on proofs as well. This proves extremely convenient.

The deduction theorem is a *metatheorem* because it is a theorem *about* propositional calculus, not a theorem *in* propositional calculus. In order to prove the deduction theorem, we can either use informal mathematical arguments based on the informal definitions of proof and so on given earlier, or we can formalize propositional calculus within some more powerful formal system and do the proof within that system. In either case, the proof is by induction on the length of the derivation of ψ from S, ϕ.

Using the deduction theorem, the proof of $\vdash (P \supset P)$ becomes trivial:

1. $P \vdash P$ Definition of proof
2. $\vdash (P \supset P)$ Deduction Theorem$[S \leftarrow \emptyset, \phi \leftarrow P, \psi \leftarrow P]$.

But even with additions such as these, it seems clear that it will not be easy or natural to develop formal proofs using the formulation of propositional calculus that we have seen so far. This is not surprising, for that formulation is a "Hilbert-style" calculus designed for metamathematical convenience—that is, it is chosen to allow short statements of notions such as "interpretation," and relatively simple proofs of soundness and completeness. It is not, however, designed for actually *doing* proofs. For that purpose, alternative formulations due to Gentzen are much more convenient.

Gentzen's *natural deduction* and *sequent calculus* formulations use fewer axioms and more rules of inference than Hilbert-style formulations, and allow more "natural" proofs.[14] Of Gentzen's two formulations, natural deduction is quite convenient

[13] This is not to say that errors in using propositional calculus are not common in formal methods, only that the line by line reasoning of a formal proof is seldom needed to detect or explain them.

[14] Sources used for this section include Barwise [Bar78b], Gallier [Gal86], and Sundholm [Sun83].

on the (hand)written page or whiteboard, but the sequent calculus is much to be preferred for computer-assisted activity since all the information relevant to the local part of a proof is gathered together in a convenient way. A *sequent* is written the form $\Gamma \rightarrow \Delta$[15] where Γ and Δ are lists of sentences called the *antecedents* and *succedents*, respectively. The intuitive interpretation is that the conjunction of the antecedents should imply the disjunction of the succedents. Equivalently, we want to show that one (or more) of the succedents is true, or that one (or more) of the antecedents is false.

A proof in sequent calculus is a tree of sequents, whose root (at the bottom) is a sequent of the form $\rightarrow \phi$ (i.e., with empty antecedent), where ϕ is the sentence to be proved. The proof tree is "grown" upwards from the root by applying *inference rules* of the form

$$\frac{\Gamma_1 \rightarrow \Delta_1 \quad \cdots \quad \Gamma_n \rightarrow \Delta_n}{\Gamma \rightarrow \Delta}\ \textbf{N}.$$

This says that the rule named N takes a leaf node of a proof tree of the form $\Gamma \rightarrow \Delta$, and adds n new leaves of the form given (n may be zero, in which case that branch of the proof tree terminates). The goal is to apply rules of inference to construct a proof tree whose branches are all terminated. This is sometimes called "backwards" proof, since we start with the conclusion to be proved. (Natural deduction, on the other hand, is an example of a "forwards" approach to proof in which we start with assumptions and work towards the conclusion.) Although the proof is *constructed* "backwards," it can subsequently be *read* "forwards."

I now give the inference rules for the propositional sequent calculus with the operators $\wedge, \vee, \supset$, and $\neg$ (omitting the "structural rules," which simply say that the order of sentences within an antecedent or succedent is unimportant, and that sentences can always be deleted from the antecedent or succedent). In the following, upper case greek letters Γ and Δ represent lists of sentences, and lower case letters ϕ, and ψ represent individual sentences. I will often write sequents in a form such as $\Gamma, \phi \supset \psi \rightarrow \Delta$, meaning that the antecedents consist of some (possibly empty) list Γ and a sentence of the form $\phi \supset \psi$.

The *Propositional Axiom* rule is the one that terminates a branch of the proof tree. It applies when the same sentence appears in both the antecedent and succedent.

$$\frac{}{\Gamma,\ \phi\ \rightarrow\ \phi,\ \Delta}\ \textbf{Ax}.$$

This rule can be seen to be plausible by thinking of the sequent as expressing the tautology $(\Gamma \wedge \phi) \supset (\phi \vee \Delta)$ where the lists Γ and Δ are interpreted as the

[15] $\Gamma \vdash \Delta$ is also used, but this may be confusing here.

conjunction and disjunction, respectively, of their elements. Equivalently, under the interpretation that we need to show that one (or more) of the succedents is true, or that one (or more) of the antecedents is false, the propositional axiom rule can be justified by noting that ϕ appears as both an antecedent and a succedent and must be either true or false.

There are two rules for each of the propositional connectives, corresponding to the antecedent and succedent occurrences of these connectives. The negation rules simply state that the appearance of a sentence in the antecedent is equivalent to the appearance of its negated form in the succedent, and vice-versa.

$$\frac{\Gamma \rightarrow \phi, \Delta}{\Gamma, \neg\phi \rightarrow \Delta}\ \neg\rightarrow \qquad \frac{\Gamma, \phi \rightarrow \Delta}{\Gamma \rightarrow \neg\phi, \Delta}\ \rightarrow\neg.$$

The inference rules $\neg \rightarrow$ and $\rightarrow \neg$ are often called the rules for "negation on the left" and "negation on the right," respectively. Similar naming conventions are used for the other rules. One way to see that the rule $\neg \rightarrow$, say, is reasonable is to consider the sequent above the line as $\Gamma \supset \phi \vee \Delta$, which is equivalent to $\neg\Gamma \vee (\phi \vee \Delta)$, which is equivalent to $(\neg\Gamma \vee \phi) \vee \Delta$, which is equivalent to $\neg(\neg\Gamma \vee \phi) \supset \Delta$, which is equivalent to $(\Gamma \wedge \neg\phi) \supset \Delta$, which is an interpretation of the sequent below the line.[16] Equivalently, under the interpretation that we need to show that one (or more) of the succedents is true, or that one (or more) of the antecedents is false, the negation rules can be seen as simply trading demonstration of the truth of ϕ for that of the falsity of $\neg\phi$, and vice-versa.

The rule for conjunction on the left is a consequence of the fact that the antecedents of a sequent are implicitly conjoined; the rule for conjunction on the right causes the proof tree to divide into two branches, in which we separately prove each of the two conjuncts.

$$\frac{\phi,\ \psi,\ \Gamma \rightarrow \Delta}{\phi \wedge \psi,\ \Gamma \rightarrow \Delta}\ \wedge\rightarrow \qquad \frac{\Gamma \rightarrow \phi,\ \Delta \qquad \Gamma \rightarrow \psi,\ \Delta}{\Gamma \rightarrow \phi \wedge \psi,\ \Delta}\ \rightarrow\wedge.$$

The rules for disjunction are "duals" of those for conjunction.

$$\frac{\phi,\ \Gamma \rightarrow \Delta \qquad \psi,\ \Gamma \rightarrow \Delta}{\phi \vee \psi,\ \Gamma \rightarrow \Delta}\ \vee\rightarrow \qquad \frac{\Gamma \rightarrow \phi,\ \psi,\ \Delta}{\Gamma \rightarrow \phi \vee \psi,\ \Delta}\ \rightarrow\vee.$$

The rule for implication on the right is a consequence of the implication "built in" to the interpretation of a sequent; the rule for implication on the left splits the proof into two branches: on one we must prove the antecedent to the implication, and on the other we may assume the consequent.

[16] In this chain of equivalences, I used the identities $(\phi \supset \psi) \equiv (\neg\phi \vee \psi)$, and one of De Morgan's Laws.

$$\frac{\psi,\ \Gamma \to \Delta \qquad \Gamma \to \phi,\ \Delta}{\phi \supset \psi,\ \Gamma \to \Delta}\ \supset\!\to \qquad\qquad \frac{\Gamma,\ \phi \to \psi,\ \Delta}{\Gamma \to \phi \supset \psi,\ \Delta}\ \to\!\supset.$$

Notice how the rules of the sequent calculus have an intuitively plausible character, and entirely symmetric construction. This makes proof construction relatively systematic and natural. To see how it works, consider a proof of the sentence

$$(P \supset Q \supset R) \supset (P \wedge Q \supset R).$$

(Note, the implies symbol $\supset$ associates to the right and binds less tightly than $\wedge$, so this sentence is actually an instance of the law of exportation.)

First we construct the goal sequent

$$\to (P \supset Q \supset R) \supset (P \wedge Q \supset R).$$

Then we must seek an inference rule whose part "below the line" has an instance equal to this sequent. There is only one choice in this case, since the sole top level operator is an implication on the right. Hence, we apply the rule for implication on the right (with $[\phi \leftarrow (P \supset Q \supset R), \psi \leftarrow (P \wedge Q \leftarrow R)]$ and Γ and Δ empty):

$$\frac{(P \supset Q \supset R) \to (P \wedge Q \supset R)}{\to (P \supset Q \supset R) \supset (P \wedge Q \supset R)}\ \to\!\supset.$$

Now we focus our attention on the sequent above the line:

$$(P \supset Q \supset R) \to (P \wedge Q \supset R)$$

and look for a rule whose part below the line can be made to match this sequent. There are two choices here: implication on the left or implication on the right. The former rule will cause the proof tree to branch and it is usually best to put this off as long as possible. So we again choose implication on the right (this time with $[\Gamma \leftarrow (P \supset Q \supset R), \phi \leftarrow P \wedge Q, \psi \leftarrow R]$ and Δ empty):

$$\frac{(P \supset Q \supset R), (P \wedge Q) \to R}{(P \supset Q \supset R) \to (P \wedge Q \supset R)}\ \to\!\supset.$$

Again we focus on the sequent above the line

$$(P \supset Q \supset R), (P \wedge Q) \to R$$

and seek an applicable rule. There are two candidates: implication on the left, or conjunction on the left. The former rule will cause the proof tree to branch, so we use conjunction on the left:

$$\frac{(P \supset Q \supset R), P, Q \to R}{(P \supset Q \supset R), (P \wedge Q) \to R}\ \wedge\to$$

Now the sequent above the line is

$$(P \supset Q \supset R), P, Q \rightarrow R$$

and we have no choice but to use the rule for implication on the left:

$$\frac{(Q \supset R), P, Q \rightarrow R \qquad P, Q \rightarrow P, R}{(P \supset Q \supset R), P, Q \rightarrow R} \supset\!\rightarrow$$

The right-hand branch can be closed immediately:

$$\frac{}{P, Q \rightarrow P, R}\ \mathbf{Ax}.$$

The left-hand branch requires another application of the rule for implication on the left:

$$\frac{R, P, Q \rightarrow R \qquad P, Q \rightarrow Q, R}{(Q \supset R), P, Q \rightarrow R} \supset\!\rightarrow.$$

Its left and right branches can then be closed:

$$\frac{}{R, P, Q \rightarrow R}\ \mathbf{Ax}$$

$$\frac{}{P, Q \rightarrow Q, R}\ \mathbf{Ax}$$

Since all the branches are closed, the theorem is proved.

We can collect all the steps together into the following representation of the proof tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{R, P, Q \rightarrow R}\ \mathbf{Ax} \qquad \dfrac{}{P, Q \rightarrow Q, R}\ \mathbf{Ax}}{(Q \supset R), P, Q \rightarrow R} \supset\!\rightarrow \qquad \dfrac{}{P, Q \rightarrow P, R}\ \mathbf{Ax}}{(P \supset Q \supset R), P, Q \rightarrow R} \supset\!\rightarrow.}{(P \supset Q \supset R) \rightarrow (P \wedge Q \supset R)} \rightarrow\!\supset}{\rightarrow (P \supset Q \supset R) \supset (P \wedge Q \supset R)} \rightarrow\!\supset$$

Notice how we were able to develop this proof in an entirely "logical" and orderly way. The reader might want to compare this proof with one using the Hilbert-style formulation of the propositional calculus introduced at the beginning of Section A.3.

A.4 Predicate Calculus

Propositional calculus is a very limited formal system and allows us to express only very simple ideas. For example [Dav89, page 43], we cannot express the idea "if x is even, then $x + 1$ is odd" in propositional calculus. To see this, suppose we try and split the idea up as two propositions: $\boxed{x}_e$ will mean "x is even" and $\boxed{x+1}_o$ will mean "$x + 1$ is odd." Then it might seem that we could express the intended idea with the compound proposition $\boxed{x}_e \supset \boxed{x+1}_o$. But this does not work: $\boxed{x}_e$ and $\boxed{x+1}_o$ are independent propositions without internal structure (which is why I have written them in boxes), whereas the sense we want to convey requires that both refer to the same x. The *predicate calculus* (also called *first-order logic* or *elementary logic* or *quantification theory*) is an extension to propositional calculus that gives us *individual variables* and the expressive power we need to make this and very many other statements concerning mathematics and the world.

The formal system of the basic predicate calculus has no nonlogical part. Its logical symbols include the propositional connectives and the *quantifiers* "for all" (also called the *universal* quantifier and written $\forall$) and "there exists" (also called the *existential* quantifier and written $\exists$). The other symbols include *constants* (generally written as lowercase letters near the front of the alphabet: $a, b, \ldots$), *variables* (generally written as lowercase letters near the end of the alphabet: $\ldots, y, z$), *function symbols* (generally written as the lowercase letters: $f, g, \ldots$), and *predicate symbols* (generally written as uppercase letters: $P, Q, \ldots$).

The grammar of predicate calculus defines its language as follows:

- A *term* is a constant symbol, a variable symbol, or a *function application* $f(\sigma, \tau, \ldots, \nu)$ where f is a function symbol and $\sigma, \tau, \ldots, \tau$ are terms. (Actually, the symbols $\sigma, \tau, \ldots, \tau$ are metavariables *representing* terms). For convenience, I often write function applications using infix notation, such as $2 + 3$, rather than $+(2, 3)$.

- An *atomic formula* has the form $P(\sigma, \tau, \ldots, \nu)$ where P is a predicate symbol and $\sigma, \tau, \ldots, \nu$ are terms. (As before, $\sigma, \tau, \ldots, \tau$ are actually metavariables *representing* terms). Computer scientists may find it more familiar to think of predicates as functions that return truth values (Russell and other early writers actually called them "propositional functions"). For convenience, I often write atomic formulas using infix notation, such as $2 < 3$, rather than $< (2, 3)$.

  There are obvious consistency constraints on the sentences we should write: if f appears somewhere applied to two arguments (e.g., in a term of the form

$f(x, y)$) we should not use it somewhere else applied to fewer or greater numbers of arguments. The same applies to predicates.[17]

- A *well-formed formula* (wff) is an atomic formula, or a series of wffs combined using the propositional connectives, or an expression of the form $(\forall x : \phi)$ or $(\exists x : \phi)$ where x is a variable symbol and ϕ is a metavariable representing a wff.

  The *scope* of a quantifier is the wff to which it applies (i.e., the one following the colon). For example, in

  $$(\forall x : even(x) \supset odd(x + 1)) \tag{A.1}$$

  the scope of the $\forall$ is the expression $even(x) \supset odd(x + 1)$. A variable is *bound* if it appears in an expression in the scope of a quantifier naming that variable; otherwise it is *free*. A wff is *open* if it contains free variables, otherwise it is *closed*.

- The *sentences* of the predicate calculus are the closed wffs. By convention, open wffs are usually interpreted as sentences by automatically providing outer levels of universal quantification to bind any free variables (this is called taking the *universal closure*). Thus the open wff $(\exists y : y = x + 1)$ is closed by interpreting it as $(\forall x : (\exists y : y = x + 1))$.[18]

Numerous syntactic shorthands are generally applied to make predicate calculus sentences more readable. For example, in the examples above, I used *even* and *odd* as predicate symbols, and used infix $=$ to denote a predicate symbol, infix $+$ to denote a function symbol, and 1 to denote a literal constant. In the most primitive predicate calculus notation, we might have to write (A.1) as

$$(\forall x : E(x) \supset O(s(x)))$$

where s is intended to indicate the successor function on the integers, and E and O are meant to indicate the "even" and "odd" predicates, respectively. No matter how they are written, this example assumes that numbers, together with certain functions (e.g., $+$ or s) and predicates (e.g., *odd* or O) on them, are part of the formal system concerned. In fact, these concepts are *not* part of the predicate calculus, but must be axiomatized or defined within it. I examine how this can

[17] To be really thorough, we should distinguish different sets of n-ary function and predicate symbols, for each n.

[18] Notice the difference between $(\forall x : (\exists y : y = x + 1))$ and $(\exists x : (\forall y : y = x + 1))$. The former says that for each x, there is some y such that $y = x + 1$—this is obviously valid. The latter says that there is an x such that, for every y, $y = x + 1$, i.e., x is one less than every y—which is obviously unsatisfiable (in the usual interpretation of arithmetic).

be done in later sections; first I examine deduction and interpretation in the basic predicate calculus.

In a Hilbert-style formulation, the axiom schemes and rules of inference for predicate calculus are those of propositional calculus (whose metavariables are now to be replaced by formulas of predicate calculus, rather than by proposition symbols, in order to construct axioms and rule instances), together with the two axiom and two rule schemes shown below. In these schemes, the notation $\phi(x)$ means a formula ϕ in which x is a free variable, and $\phi[x \leftarrow t]$ means the substitution instance of ϕ with all free occurrences of x replaced by the term symbolized by t.[19]

The following axiom schemes are added to those of propositional calculus given on page 230:

A4. $(\forall x : \phi(x)) \supset \phi[t \leftarrow x]$, provided no free occurrences of x in ϕ lie in the scope of any quantifier for a free variable appearing in the term t (we say that t is free for x in ϕ).[20] This axiom scheme simply says that if some formula ϕ is true for all x, then it is certainly true when some particular term t is substituted for x in ϕ (provided no free variables get "captured").

A5. $\phi[x \leftarrow t] \supset (\exists x : \phi(x))$, provided t is free for x in ϕ. This axiom scheme says that we can conclude that there exists some x satisfying the formula ϕ if some substitution instance of ϕ is true.[21]

The following rule schemes of *generalization* are added to those of propositional calculus:

$$\frac{\psi \supset \phi(v)}{\psi \supset (\forall x : \phi(x))},$$

and

$$\frac{\phi(v) \supset \psi}{(\exists x : \phi(x)) \supset \psi}$$

where the variable v is not free in ψ. The rule of universal generalization can best be understood by considering the simpler case where ψ is true. Then the rule becomes

$$\frac{\phi(v)}{(\forall x : \phi(x))}$$

[19] Mathematicians usually indicate a substitution instance by $\phi[t/x]$ rather than $\phi[x \leftarrow t]$. I prefer my notation because the arrow makes it easy to remember the direction of the substitution. Substitution is an operation in the object language (it transforms one formula into another). I have also been using the notation $A1[\phi \leftarrow P]$ to indicate the metalevel operation in which an axiom or rule instance is constructed by letting ϕ be P in the scheme $A1$. This ambiguous notation is nonstandard, but I find it convenient.

[20] This caveat is called the "occurs check." To see why it is necessary, let ϕ be $(\exists y : x + 1 = y)$ and let t be y. Prolog interpreters ignore the occurs check in the interest of speed, and are unsound for this reason among others [NS93, Chapter III].

[21] Some formulations include only the universal quantifier; in these cases, the existential quantifier can be introduced by the definition $(\exists x : \phi(x)) \equiv \neg(\forall x : \neg\phi(x))$.

which says that if ϕ is true for some arbitrary v, then it must be true for all x. Existential generalization can be derived from universal generalization by the transformation mentioned in the previous footnote.

The models of predicate calculus are constructed as follows.[22] Given a nonempty set $\mathcal{D}$ as the domain of interpretation, an interpretation associates an n-place total function on $\mathcal{D}$ with each n-place function symbol, an n-place relation with each n-place predicate symbol, and an element of $\mathcal{D}$ to each constant symbol.[23] The interpretation of $P(t_1, t_2, \ldots, t_n)$ for predicate symbol P is true if and only if the tuple $(v(t_1), v(t_2), \ldots v(t_n))$ is in the set $v(P)$ where $v(P)$ is the relation interpreting P, and $v(t_i)$ is the interpretation of the term t_i. The interpretation of $f(t_1, t_2, \ldots, t_n)$ for function symbol f is the value $v(f)(v(t_1), v(t_2), \ldots v(t_n))$ where $v(f)$ is the function interpreting f, and $v(t_i)$ is the interpretation of the term t_i. The propositional connectives have their usual (truth-table) meaning; $(\forall x : \phi(x))$ is interpreted to be true if and only if the interpretation of ϕ is true for all values of the variable x in the domain $\mathcal{D}$, and $(\exists x : \phi(x))$ is interpreted to be true if and only if the interpretation of ϕ is true for some value of the variable x taken from the domain $\mathcal{D}$. (If ϕ does not contain the variable x, the interpretations of $(\forall x : \phi(x))$ and $(\exists x : \phi(x))$ are the same as the interpretation for ϕ.)

Like propositional calculus, predicate calculus is sound and complete (soundness is relatively straightforward; completeness was established by Gödel in his Ph.D. thesis in 1930[24]). That is, the theorems (i.e., the sentences that can be deduced using the axioms and rules of inference given above) coincide with the universally valid sentences (i.e., those that valuate to true in all interpretations).

We have now covered enough material to formalize and interpret the example that began this appendix:

> That plane is a Boeing 737;
> All Boeing planes, except the 747, have two engines;
> Therefore that plane has two engines.

[22] The definitions that follow use the terminology of set theory. An n-place *relation* on the set $\mathcal{D}$ is any set of n-tuples on $\mathcal{D}$, that is any set of objects of the form $(a_1, a_2, \ldots, a_n)$, where each a_i is a member of $\mathcal{D}$. An n-place *function* f on the set $\mathcal{D}$ can be considered as $n+1$-place relation with the property that if $(a_1, a_2, \ldots, a_n, x)$ and $(a_1, a_2, \ldots, a_n, y)$ are both members of f, then $x = y$ (i.e., the last member of the tuple is uniquely determined by the first n). In this case, the *application* of f to the *arguments* $(a_1, a_2, \ldots, a_n)$ is written $f(a_1, a_2, \ldots, a_n)$ and yields the value x. A function is *total* if it has a value for every combination of arguments.

[23] I am simplifying a bit here. Interpretations are correctly defined relative to a sequence of values from $\mathcal{D}$ that supplies arbitrary, but fixed, interpretations to free variables.

[24] Skolem had essentially demonstrated consistency some years earlier, but did not realize it [Hod83].

We can formalize this in first-order predicate calculus

$$\frac{\begin{array}{l}\texttt{is\_a\_Boeing\_plane}(p) \wedge \texttt{is\_a\_737}(p) \\ \forall x : \texttt{is\_a\_Boeing\_plane}(x) \wedge \neg\texttt{is\_a\_747}(x) \supset \texttt{has\_two\_engines}(x)\end{array}}{\texttt{has\_two\_engines}(p)}$$

where `is_a_Boeing_plane`, `is_a_737`, `is_a_747`, and `has_two_engines` are predicates, p is a constant, and x is a variable. A little thought shows that this deduction is not valid: we also need another premise to state that 737s and 747s are different. The following is adequate for our purposes:

$$\forall x\colon \texttt{is\_a\_737}(x) \supset \neg\texttt{is\_a\_747}(x).$$

Then, applying the logical axioms and rules of inference given earlier (use A4 to instantiate the two universally quantified premises with $[x \leftarrow p]$, then use propositional reasoning), we can prove the conclusion from the three premises. Since predicate calculus is sound, this tells us that the conclusion will be true in any interpretation that valuates the premises to true. Notice, however, that if we substitute `is_a_727` for `is_a_737`, we would still have a valid proof, but a conclusion that is false when the domain $\mathcal{D}$ of interpretation is "airplanes in current service" and the predicates have their intuitive meaning. How can this be? The explanation is that the second premise

$$\forall x : \texttt{is\_a\_Boeing\_plane}(x) \wedge \neg\texttt{is\_a\_747}(x) \supset \texttt{has\_two\_engines}(x)$$

is false in this interpretation. But if that is so, then our original conclusion (about the 737) is also suspect in this interpretation. And indeed it is: although the conclusion happens to be true in this interpretation, and although the line of reasoning is valid (so that the conclusion must be true in any interpretation that validates the premises), one of the premises is false in the intended interpretation. This example illustrates a central issue in formal methods: because we intend to make use of any conclusion we prove, we must have a particular interpretation in mind and we need to pay great attention to validating our premises in that interpretation.

A.4.1 Skolemization and Resolution

Unlike propositional calculus, predicate calculus is only semidecidable. Because it introduces some concepts and terminology that are useful in understanding certain theorem proving strategies, I will sketch one of the ways to demonstrate semidecidability of predicate calculus. Recall that semidecidable means there must be some computer program that is guaranteed to terminate with the answer "yes" when given a theorem; it is not required to terminate when given a nontheorem (but if it does, it must give the answer "no"). Because the predicate calculus is sound and

complete, the desired semidecision procedure could either be proof theoretic (i.e., it could look for a proof of the given sentence), or model theoretic (i.e., it could attempt to show that the sentence is universally valid). The approach I will follow is model theoretic.

Suppose we have to establish a sentence of the form $(\forall x : \phi(x))$; to make things concrete, consider $(\forall x : x < x + 1)$. If this sentence is valid, then any particular substitution instance, say $17 < 17{+}1$, must be true. So one way to prove the sentence might be to choose some substitution instance such as this "at random" and check that one instance. The problem, of course, is that a particular, interpreted constant such as 17 might have some special properties, and we might just happen to pick one that satisfied the sentence, whereas another might not (e.g., $(\forall x{:}\, x \times x = x)$ might appear valid if it were enough to exhibit $[x \leftarrow 1]$). But if instead of a particular numeral, we choose a new symbolic constant, say c, that appears nowhere else and could still establish $c < c{+}1$, then we would surely have done enough to establish the original sentence. Reverting to the general case, the idea is to establish $(\forall x : \phi(x))$ by establishing $\phi(c)$ for some new constant c. Notice that $\phi(c)$ has no quantifiers nor variables: it is a *propositional* sentence and therefore, by the decidability of the propositional calculus, we can decide its validity. We seem to have reduced the decision problem for universally quantified formulas of predicate calculus to that of propositional calculus.

Now consider a slightly more complicated sentence: $(\forall x : (\exists y : y = x + 1))$. To establish this, we can start by substituting some arbitrary constant, say c, for the universally quantified variable x to yield the simpler sentence $(\exists y : y = c{+}1)$. Then, to establish this existential sentence it is enough to find some constant term that can be substituted for y in order to make $y = c + 1$ true (obviously, $[y \leftarrow c + 1]$ does the job here). The quantifier-free formula $y = c + 1$ is called a *Skolem form* of the original formula $(\forall x : (\exists y : y = x + 1))$. A substitution instance of the Skolem form, in which all its free variables are replaced by terms involving only constant symbols, is called a *ground instance* of the original formula. The idea behind the process I am developing is that the original first-order sentence must be valid if some ground instance is propositionally valid (i.e., a tautology). Now the construction of the Skolem form is a mechanical process (I will explain it shortly), and propositional validity is decidable—so to test the validity of a first-order formula, all we have to do is generate the Skolem form, and then search for a ground instance that is a tautology.[25] If the original formula is valid, then eventually our search will terminate, but if it is invalid, then our search may go on forever, generating more and more complicated ground instances without ever finding one that is a tautology.

[25] This is not quite accurate: as explained shortly, we may need a disjunction of ground instances to obtain a tautology.

I have just given a crude sketch of the semidecidability of first-order predicate calculus.[26] This idea can be exploited in mechanical "theorem provers" (I use quotes because this is a model-, not proof-theoretic approach)—although it is usual to employ heuristics, or user-supplied hints, to generate ground instances rather than exhaustive search.[27] In order to use this approach, it is necessary to understand Skolemization (as the process of constructing the Skolem form is called) in a little more detail.

My simple example might have made it appear that to Skolemize a first-order sentence, all we have to do is replace the universally quantified variables with (different) arbitrary constants, leave the existentially quantified variables alone, and then remove all the quantifiers. If we apply this naive scheme to the (obviously false) sentence $(\exists x{:}\,\phi(x)) \supset (\forall y{:}\,\phi(y))$ we would obtain $\phi(x) \supset \phi(c)$, which has a valid ground instance under the substitution $[x \leftarrow c]$. We have just "proved" a false theorem, so something must have gone wrong.

Our mistake was that we did not take care of the "implicit" negation in the antecedent to an implication. Since $a \supset b$ is the same as $\neg a \vee b$, the original sentence could have been written as $\neg(\exists x{:}\,\phi(x)) \vee (\forall y{:}\,\phi(y))$, and this is equivalent to $(\forall x{:}\,\neg\phi(x)) \vee (\forall y{:}\,\phi(y))$. Our Skolemization procedure (correctly) transforms this to $\neg\phi(d) \vee \phi(c)$, which is not a tautology. The problem, then, is that existential quantifiers in the antecedent to an implication are "essentially" universal (and vice-versa). We see that Skolemization has to replace the *essentially universally* quantified variables with different arbitrary constants, and to leave the *essentially existential* ones alone. To determine the essential "parity" of a quantifier, simply count the number of negations in whose scope it occurs, noting the implicit negation in the antecedent of an implication (and expanding equivalences of the form $\phi \equiv \psi$ as two implications: $\phi \supset \psi \wedge \psi \supset \phi$); if a quantifier appears within an odd number of negations, then its essential parity is the opposite of its appearance (i.e., an existential is essentially universal, and vice-versa).

Even with this adjustment, our Skolemization process is still flawed. Consider the example we looked at earlier, but with the order of the quantifiers reversed: $(\exists y : (\forall x : y = x + 1))$. This sentence is unsatisfiable (it says there is some y which is equal to $x + 1$ for every x, whereas the original, valid, sentence said that for every x, there is some y that is equal to $x + 1$). Yet our modified, but still naive, Skolemization algorithm produces exactly the same Skolem form, namely $y = c + 1$, as for the original sentence.

[26] My examples used predicate calculus enriched with arithmetic; this is also semidecidable, provided certain restrictions are placed on the fragment of arithmetic employed. This is discussed later in Section A.5.2.

[27] If the search is not exhaustive, the process is no longer a semidecision procedure: if a valid ground instance is found, then the original sentence is certainly valid, but there is no guarantee that the search will find such an instance.

One way to better understand what is going on is to imagine the process of forming the final ground instance as a contest between ourselves, seeking to find substitutions for the essentially existential variables, and a malign opponent who chooses instances for the essentially universal variables [Hod83]. We and our opponent take turns, working from the outside quantifiers towards the innermost; our goal is reach a valid ground sentence, the opponent's goal is prevent us doing so.

If we first consider the valid sentence $(\forall x : (\exists y : y = x + 1))$ we see that the opponent plays first (since the outermost quantifier is essentially universal). The opponent might decide to substitute 17, say, for x, leaving us with $(\exists y : y = 17 + 1)$. We can now choose to substitute 18 for y, yielding $18 = 17 + 1$, so we win.

Now consider the unsatisfiable form: $(\exists y : (\forall x : y = x + 1))$. Here, we have to play first, so we choose some arbitrary constant, say 9, to leave $(\forall x : 9 = x+1)$. Now our opponent plays, and is careful to choose any number but 8—say 13—thereby winning the contest.

Notice that what happened in the last case was that our opponent did not need to choose a substitution for x until we had made a choice for y; thus, the choice made by the opponent could *depend* on our choice of y—in other words, it could be some *function* $f(y)$ of y. To perform Skolemization correctly, we may only replace essentially universal variables with constants when they are *not* in the scope of some essentially existential quantifier. When they are in the scope of essentially existential quantifiers, the universal variables must be replaced by arbitrary new *function* symbols (called *Skolem functions*) that take the existential variables as arguments. Thus, the correctly Skolemized form of $(\exists y : (\forall x : y = x + 1))$ is $y = f(y) + 1$, which has no valid ground instance. For an example of this kind that is valid, consider $(\exists y : (\forall x : y < x + 1))$, where the variables are constrained to be natural numbers. The Skolem form is $y < f(y) + 1$, which has the valid ground instance $0 < f(0) + 1$ (i.e., $[y \leftarrow 0]$). (The constraint to natural numbers ensures $0 \leq f(y)$ no matter what function is chosen for f.) Because we know nothing about the Skolem function f, it models any strategy our opponent may choose. If we can construct a valid ground instance in the presence of Skolem functions, it means we can always win the contest, no matter what the opponent's strategy.

We now know how to Skolemize first-order sentences correctly, but there is one last problem in this approach to theorem proving. To see this, consider the (valid) sentence

$$(\exists x : \phi(x)) \vee (\exists y : \psi(y)) \supset (\exists z : \phi(z) \vee \psi(z)).$$

The Skolem form of this sentence is

$$(\phi(a) \vee \psi(b)) \supset (\phi(z) \vee \psi(z))$$

where a and b are arbitrary Skolem constants, and we have to find a ground substitution for z that will make the formula a tautology. Unfortunately, it is not hard to see that no such substitution exists.

The problem is that we have to find a single substitution for z without knowing which of $\phi(a)$ or $\psi(b)$ is true (although we do know that at least one of them is true). If we knew which were true, we could find the right substitution ($[z \leftarrow a]$ if $\phi(a)$ is true, otherwise $[z \leftarrow b]$). A little thought should convince us that this is enough: no matter what the true state of affairs, there is always some satisfiable ground instance. Indeed, this is enough, and the result that justifies it is the Herbrand-Skolem-Gödel theorem,[28] which says that a first-order sentence is valid if and only if some finite *disjunction* of ground instances of its Skolem form is tautological. In the case of our example, the disjunction

$$\begin{array}{ccc} (\phi(a) \vee \psi(b)) & \supset & (\phi(a) \vee \psi(a)) \\ \vee & & \\ (\phi(a) \vee \psi(b)) & \supset & (\phi(b) \vee \psi(b)) \end{array}$$

is valid, and we have succeeded in proving the original theorem.

Almost all mechanical theorem provers employ Skolemization in one form or another, so it is worth having some familiarity with the process. Some systems convert formulas to *prenex* form (in which no quantifiers appear in the scope of a propositional connective) before Skolemizing. This is done by repeatedly using the law $(\exists x : \phi) \equiv \neg(\forall x : \neg\phi)$ and its dual, and by renaming variables if necessary so that expressions such as $(\forall x{:}\, \phi) \vee \psi$ can be rewritten as $(\forall x{:}\, \phi \vee \psi)$ (x must not occur free in ψ), and similarly for expressions involving other connectives. In an interactive system, such transformations are disadvantageous, since users are required to examine formulas presented in a very different form than those they typed in. In automatic theorem provers, however, conversion to various special forms can assist the systematic search for a proof. *Resolution* provers, for example, generally eliminate all connectives other than $\wedge$, $\vee$, and $\neg$ (by, for example, transforming $\phi \supset \psi$ into $\neg\phi \vee \psi$), convert the resulting formulas to prenex form, Skolemize them, and then convert to *conjunctive normal form* (CNF) by repeated applications of de Morgan's laws. In CNF, each formula is written as a conjunction of *clauses*, where a clause is a disjunction of *literals*, which are atomic formulas or negations of atomic formulas (the empty clause is identified with *false*).

Propositional formulas in CNF can often be shown to be unsatisfiable by repeated application of the *one-literal rule*: if a clause consists of just a single literal, then that clause can be deleted, and all instances of the negated form of that literal[29] can be deleted from all other clauses: if that deletion results in any clause becoming empty (i.e., *false*) then the original formula was unsatisfiable. For example:

[28] This result in model theory should not be confused with Herbrand's Theorem, which is a much deeper result in proof theory.

[29] If the literal is a negation, then its negated form is the unnegated atomic formula (e.g., the negated form of $\neg P$ is simply P).

$(P \vee Q \vee \neg R) \wedge (P \vee \neg Q) \wedge \neg P \wedge R \wedge S$	apply one-literal rule to $\neg P$
$(Q \vee \neg R) \wedge \neg Q \wedge R \wedge S$	apply one-literal rule to $\neg Q$
$\neg R \wedge R \wedge S$	apply one-literal rule to $\neg R$
empty $\wedge S$	we are done.

The *resolution* rule for propositional calculus extends the one-literal rule by looking for clauses that contain a *complementary pair* of literals P and $\neg P$. Such a pair of clauses, for example $(P \vee R)$ and $(\neg P \vee Q)$, are then replaced by their *resolvent* $(R \vee Q)$ (thus the one-literal rule is the special case of resolution when either R or Q is empty). Resolution is extended to first-order clauses (i.e., those containing variables) by seeking a single substitution that can be applied to two literals to make them a complementary pair, and then applying the same substitution to the resolvent. The process of constructing a substitution that will cause two literals to become a complementary pair is called *unification*, and the substitution that results is called a *unifier*. Here is an example of first-order resolution:

$(\neg S(y) \vee C(y)) \wedge S(b) \wedge V(a,b) \wedge (\neg C(z) \vee \neg V(a,z))$	$[y \leftarrow b]$ on clauses 1, 2
$C(b) \wedge V(a,b) \wedge (\neg C(z) \vee \neg V(a,z))$	$[z \leftarrow b]$ on clauses 1, 3
$V(a,b) \wedge \neg V(a,b)$	Clauses 1, 2
empty	we are done

Resolution is a complete refutation procedure for first-order logic: if a sentence is unsatisfiable, then resolution will terminate with the empty clause. For theorem proving, we simply conjoin all the premises with the *negated* conclusion, and then use resolution to test whether the result is unsatisfiable: if it is, then the original theorem is true. If the original theorem was untrue, then resolution may not terminate.

The basic resolution and unification algorithms are due to Robinson [Rob65]. A great many extensions and heuristic optimizations to resolution have been developed over the years; some of the more fundamental ones are described in the standard text by Chang and Lee [CL73]; Otter is a modern theorem prover based on highly efficient implementations of several resolution strategies [McC90]. A variety of resolution called SLD-resolution is very effective for Horn clauses (clauses that contain at most one negated literal) and is the technique underlying interpreters for the language Prolog [NS93, Chapter III].

Although resolution theorem provers can be quite effective in some domains, they are of little use in formal methods. Formal methods generally require more than just pure first-order predicate calculus (e.g., they require theories for arithmetic and various datatypes, and possibly set theory or higher-order quantification), and resolution is not at all effective in these contexts.[30] In addition, many of the theorems

[30] If we simply add the axioms for the theories concerned, then the search space becomes so huge that resolution seldom terminates in reasonable time. If we add decision procedures for the theories concerned, then unification needs to be performed modulo these interpreted theories and it is a research topic to make this work effectively [Sti85].

we try to prove in formal methods are untrue when first formulated, and resolution provides little help in such cases (though it can sometimes generate counterexamples). And for true theorems, resolution simply affirms their truth: it does not assist us to develop an argument that can be subjected to human review. Nonetheless, some of the ideas from resolution find application in almost all modern theorem provers. In particular, unification is a fundamental technique for creating substitution instances, and highly efficient (i.e., linear-time) unification algorithms have been developed, as have extensions to the higher-order case.

A.4.2 Sequent Calculus

The alternatives to model theoretic approaches to "theorem proving," such as that sketched in the previous section, are proof theoretic approaches. These include the first-order sequent calculus, which is obtained by extending the propositional sequent calculus of Section A.3.1 with the following rules.

There are four "quantifier rules." In the $\rightarrow \forall$ and $\exists \rightarrow$ rules (i.e., those on the top right and bottom left), a must be a new constant[31] (these rules are the analogs of Skolemization).

$$\frac{\Gamma,\ A[x \leftarrow t] \rightarrow \Delta}{\Gamma,\ (\forall x\colon A) \rightarrow \Delta}\ \forall \rightarrow \qquad \frac{\Gamma \rightarrow A[x \leftarrow a],\ \Delta}{\Gamma \rightarrow (\forall x\colon A),\ \Delta}\ \rightarrow \forall$$

$$\frac{\Gamma,\ A[x \leftarrow a] \rightarrow \Delta}{\Gamma,\ (\exists x\colon A) \rightarrow \Delta}\ \exists \rightarrow \qquad \frac{\Gamma \rightarrow A[x \leftarrow t],\ \Delta}{\Gamma \rightarrow (\exists x\colon A),\ \Delta}\ \rightarrow \exists$$

We also need a rule for terminating a branch of the proof tree when we encounter a nonlogical axiom or a previously-proved lemma among the consequents of a sequent:

$$\frac{}{\Gamma \rightarrow \phi,\ \Delta}\ \textbf{Nonlog-ax}$$

where ϕ is a nonlogical axiom or previously-proved lemma.

For convenience in developing proofs, it is useful to provide an additional rule called "cut." This can be seen as a mechanism for introducing a case-split or lemma into the proof of a sequent $\Gamma \rightarrow \Delta$ to yield the subgoals $\Gamma, \phi \rightarrow \Delta$ and $\Gamma \rightarrow \phi, \Delta$. These are equivalent to assuming ϕ along one branch and having to prove it on the other.[32]

[31] Actually, a constant that does not occur in the consequent of the sequent. This constraint is called the *eigenvariable condition*.

[32] Alternatively, applying the rule for negation on the right, this can be seen as equivalent to assuming ϕ along one branch and $\neg\phi$ along the other.

$$\frac{\phi,\ \Gamma \rightarrow \Delta \qquad \Gamma \rightarrow \phi,\ \Delta}{\Gamma \rightarrow \Delta}\ \textbf{Cut.}$$

The Cut Elimination Theorem (also known as Gentzen's Hauptsatz) is one of the most famous results in proof theory: it says that any derivation involving the cut rule can be converted to another (possibly much longer one) that does not use cut. Since cut is the only rule in which a formula (ϕ) appears above the line that does not also appear below it, it is the only rule whose use requires "invention" or "insight"; thus the cut elimination theorem provides the foundation for another demonstration of the semi-decidability of the predicate calculus.

A.5 First-Order Theories

First-order theories are simply those that add some nonlogical axioms (and possibly rules of inference) to the predicate calculus. In this section, I consider four very important classes of first-order theories: equality, arithmetic, simple computer science datatypes (such as lists, trees etc.), and set theory.

All of these are very useful in computer science, and any system intended to support formal methods should normally provide all four.

A.5.1 Equality

The first-order theory of equality (or "identity" as it is sometimes called) simply adds a distinguished two-place predicate (generally called *equals* and written as infix $=$) to the symbols of the predicate calculus. The fundamental idea of equality is that $x = y$ if and only if "anything that may be said of x may be said of y, and vice-versa."[33] This idea of the "identity of indiscernibles" was first stated explicitly by Leibniz. In more technical terms, if some property ϕ holds for x, and $x = y$, then ϕ should also hold when y is substituted for some or all instances of x in ϕ. This can be formalized as follows:

Leibniz' law: $(\forall x, y : x = y \supset \phi \supset \phi[\![x \leftarrow y]\!])$[34]

where I use $\phi[\![x \leftarrow y]\!]$, rather than $\phi[x \leftarrow y]$, to indicate that only some instances of x need be replaced by y.

To get things started, we assert that everything equals itself:

reflexivity: $(\forall x : x = x)$.

[33] Most of this material is drawn from Tarski [Tar76].

[34] The symbol $=$ binds tighter than the propositional connectives.

From these axiom schemes, it is possible to deduce that $=$ satisfies the properties of symmetry and transitivity in addition to reflexivity, and is therefore an equivalence relation.

symmetry: $(\forall x, y : x = y \supset y = x)$,

transitivity: $(\forall x, y, z : x = y \wedge y = z \supset x = z)$.

We can also deduce that $=$ satisfies the property of substitutivity:

substitutivity: for any term or atomic formula ϕ, $x = y \supset \phi = \phi[x \leftarrow y]$

which is what distinguishes true equality from a mere equivalence relation: it says that we can always substitute equals for equals.

Sometimes the quantifier $\exists!$ (pronounced "there exists exactly one") is added to first-order theories with equality. It is defined by

$$(\exists! x : \phi(x)) \equiv (\exists x : \phi(x)) \wedge (\forall x, y : \phi(x) \wedge \phi(y) \supset x = y).$$

Notice that the first conjunct on the right-hand side expresses "at least one," while the second expresses "at most one."

The propositional connective $\equiv$ (equivalence, or "if and only if") satisfies Leibniz' law, and therefore functions as an equality relation on wffs. This means that certain sentences can be proved by applying the rules for equality reasoning to $\equiv$. For example, $(x \equiv y) \supset (x \supset y)$ can be proved from Leibniz' law (let ϕ be x). When this approach can be used, it is fast and simple and generally to be preferred to reasoning from the propositional properties of the connective. However, $\equiv$ is more than an equality relation, so equality reasoning does not capture all its properties and it is sometimes necessary to revert to propositional reasoning on $\equiv$. For example, $(\neg x \vee y) \equiv (x \supset y)$ requires propositional reasoning.


When constructing models for a first-order theory with equality, we usually want the interpretation of "=" to be the identity relation on the domain of the interpretation. Such models are called *normal* and it is possible to show that a first-order system with equality has a model if and only if it has a normal model. Thus, nothing is lost (or gained) by restricting attention to normal models.


Often, we will also want our models to have "no confusion" and "no junk." Suppose we have a theory with just two axioms: $a = b$ and $c = d$ (where a, b, c, and d are constants). Then we could have a model in which the domain of interpretation has but a single element, and all four constants are assigned to that single element. The equation $a = c$ will be true in this model, but cannot be

proved from our axioms: the model makes too many things equal (it has confusion). Alternatively, we could have a model with six members m_a, m_b, m_c, m_d, m_x, and m_y. The constants of our theory might be assigned to the first four, leaving the last two as "junk" superfluous to our needs. *Initial* models are, roughly speaking, those that have enough elements so that those assigned to terms that the axioms do not require to be equal can, in fact, be distinct (no confusion), yet with no elements left over (no junk) [GTWW77].

A.5.1.1 Sequent Calculus Rules for Equality

These rules directly encode the axiom schemes of reflexivity and Leibniz' law.

$$\frac{}{\Gamma \to a = a,\ \Delta}\ \textbf{Refl} \qquad \frac{a = b,\ \Gamma[\![a \leftarrow b]\!] \to \Delta[\![a \leftarrow b]\!]}{a = b,\ \Gamma \to \Delta}\ \textbf{Repl}$$

Observe that the rule **Refl** is an instance of the rule **Nonlog-ax** given in the previous section.

A.5.1.2 Rewriting and Decision Procedures for Equality

Reasoning about equality is so fundamental that most theorem provers provide special treatment for it. For example, chains of equality reasoning such as that required to prove the following theorem arise frequently in formal methods:

$$i = j \wedge k = l \wedge f(i) = g(k) \wedge j = f(j) \wedge m = g(l) \supset f(m) = b(k).$$

Trying to prove formulas such as this by repeated application of Leibniz' law, or the derived axioms and inference rules given earlier, soon becomes hopelessly inefficient as the number or size of the formulas increases. A very efficient method for reasoning with ground equalities of this kind is based on *congruence closure* [DST80, Sho78b].

Equations also commonly arise in the form of definitions, such as that for the absolute value function:

$$|x| = \textbf{if } x < 0 \textbf{ then } -x \textbf{ else } x.$$

One way to prove a theorem such as $|a+b| \leq |a| + |b|$ is to expand the definitions and then perform propositional and arithmetic reasoning. "Expanding a definition" means finding a substitution for the left hand side of the definition that will cause it to match a term in the formula (e.g., $[x \leftarrow a + b]$ will match $|x|$ with $|a + b|$),

and then replacing that term by the corresponding substitution instance of the right hand side of the definition concerned—for example,

$$|a+b| = \textbf{if } a+b<0 \textbf{ then } -(a+b) \textbf{ else } a+b.$$

Expanding definitions is a special case of the more general technique of *rewriting*, which can be used with arbitrary equations provided the free variables appearing on the right hand side of each equation are a subset of those appearing on its left. The idea is to find substitutions for the free variables in the left hand side of an equation that will cause it to match some part of the formula being proved; that part is then replaced ("rewritten") by the corresponding substitution instance of the right hand side of the equation concerned. The process of finding substitutions for the free variables in the left hand side by trying to make it equal some subexpression in the formula of interest is called *matching*; it is similar to unification, except that we only seek substitutions for the variables in the equation being matched, and not for the variables in the expression it is being matched against (hence, matching is sometimes described as "one way" unification). Notice that although equations are symmetric (i.e., they mean the same whether written as $a=b$ or as $b=a$), rewriting gives them a left-to-right orientation; when viewed in this way, they are generally called *rewrite rules* rather than equations.

Selection and orientation of equations to be used as rewrite rules, and identification of the target locations where rewriting is to be performed, can be undertaken either by the user or by some automated strategy. One strategy is to rewrite whenever it is possible to do so. Unfortunately, this process may not terminate; a set of rewrite rules is said to have the *finite termination* property if rewriting does always terminate.[35] Often there will be two or more opportunities for rewriting in a given expression; if the final result after rewriting to termination is independent of the choices made at each step, then the set of rewrite rules is said to have the *unique termination* (also known as Church-Rosser) property. If a theory can be specified by a set of rewrite rules with the finite and unique termination properties, then rewriting can serve as a decision procedure for the theory concerned: to decide whether two terms are equal, simply rewrite them to termination; if the results are syntactically identical, then the original terms were equal (i.e., rewriting to termination yields a normal form). This procedure is sound, and for ground formulas it is complete. We must be careful, however, if we wish to deduce disequality. Suppose,

[35] Rules that express commutativity (e.g., $x+y=y+x$) can be applied repeatedly (e.g., $a+b \rightarrow b+a \rightarrow a+b \ldots$) and so render a set of rewrite rules nonterminating. This difficulty can be overcome by imposing restrictions that lead to a true normal form (e.g., only apply the rewrite if the substitution instance for x is less than that for y in some suitable ordering). Similar techniques can be used for operators that are associative (and for the combined associative-commutative case) [Sti81]. These techniques are usually embedded in the matching rather than the rewriting mechanism, and referred to as *AC-matching*.

for example, we used the rewrite rules $a \rightarrow b$ and $c \rightarrow d$ to rewrite a to b and c to d and then observed that b and d are in normal form, but not syntactically identical: may we then deduce $a \neq c$? Plainly it would not be sound to do so in the standard semantics, since we could have a model with only a single element (so that all terms are equal). However, deducing disequalities in this way is sound (and complete) for the *initial* model. Consequently, systems that use rewriting to normal form as their main (or only) means of deduction generally use the initial model semantics (OBJ [Gog89] does this), whereas systems that use rewriting as merely one method among several generally use the classical semantics and do not infer disequalities from unequal normal forms (disequality can then only be inferred from axioms that mention it explicitly).

Given an arbitrary set of equations, there are some quite effective heuristic procedures for testing for the finite and unique termination properties [DJ90]; one, known as Knuth-Bendix completion, can often extend a set of rewrite rules that does not have unique termination into one that does. Beyond ordinary rewriting is *conditional* rewriting, which is used for axioms having the form $a \supset b = c$. Various control strategies are possible: the simplest will only rewrite b to c if a can be proved, others will do the rewrite and carry a along as an additional proof obligation.

Term rewriting is so effective that it provides the main means of deduction in some systems, for example Affirm [Mus80], Larch [GH93], and RRL [KZ88]. The very powerful Boyer-Moore prover [BM79, BM88] integrates a number of techniques, but rewriting is one of the most central. Techniques similar to rewriting are used in resolution provers under the name "paramodulation" and its variants. In general, congruence closure and term rewriting are fundamental to productive theorem proving, and theorem provers lacking these mechanisms should find little application in formal methods.

A.5.2 Arithmetic

Numbers are fundamental to mathematics. Peano's arithmetic [Pea67] (much of which should really be attributed to Dedekind) is a formal system that characterizes the natural numbers (i.e., the nonnegative integers $0, 1, 2, \ldots$)[36] by adding nonlogical axioms to the predicate calculus with equality. The first four axioms introduce the constant 0, the successor function *succ* and the predicate $\mathcal{N}$, which we can read "is a number":

- $\mathcal{N}(0)$ (0 is a number),

[36] Sometimes (and, indeed, in Peano's original formulation) the natural numbers are considered to start at 1; nowadays it is more common to refer to the numbers starting from 1 as the positive integers.

- $\mathcal{N}(x) \supset \mathcal{N}(succ(x))$ (the successor of a number is a number),
- $\mathcal{N}(x) \supset succ(x) \neq 0$ (0 is not the successor of any number),
- $\mathcal{N}(x) \wedge \mathcal{N}(y) \wedge succ(x) = succ(y) \supset x = y$ (numbers with identical successors are identical).

The fifth axiom is the scheme of *mathematical induction*:

- $(\phi(0) \wedge (\forall x : \mathcal{N}(x) \wedge \phi(x) \supset \phi(succ(x)))) \supset (\forall z{:}\mathcal{N}(z) \supset \phi(z))$.

  This says that to establish that some property ϕ holds for any natural number, it is sufficient to prove that it holds for 0 and, given that it holds for some arbitrary natural number x, to prove that it also holds for $succ(x)$.

The first four axioms imply that there are infinitely many numbers; the fifth axiom ensures there are not too many.

The numerals are introduced by regarding 1 as $succ(0)$, 2 as $succ(1)$, and so on. The remaining axioms introduce addition (written as infix $+$): and multiplication (written as infix $\times$). I drop the tedious predicate $\mathcal{N}$ and simply assume that the variables range over numbers (cf. many-sorted logic in Section A.9):

- $x + 0 = x$,
- $x + succ(y) = succ(x + y)$,
- $x \times 0 = 0$,
- $x \times succ(y) = x \times y + x$.

Although Peano's system is considered sound, it is incomplete: there are valid statements in arithmetic that cannot be proved using Peano's axioms. It might seem that this could be remedied by adding more powerful axioms, but this is not so. Gödel's incompleteness theorems, probably the most famous theorems in the whole of logic, show that all reasonably powerful formal systems are necessarily incomplete. The first incompleteness theorem says that any consistent formal system that includes arithmetic must be incomplete;[37] the second says that the consistency of such a system cannot be proved within the system (i.e., the formula that asserts consistency is an example of a true but unprovable statement).[38] Profound though it is, Gödel's first incompleteness theorem is not a limitation on the decidability of

[37] The first incompleteness theorem has been formally verified [Sha94]—almost certainly the hardest mechanically checked formal verification ever undertaken.

[38] Gentzen and Gödel demonstrated the consistency of Peano Arithmetic using metamathematical arguments that cannot be formalized within Peano Arithmetic.

formulas that arise in formal methods any more than the existence of algorithmically unsolvable problems (e.g., the halting problem for Turing machines) is a limitation on the practical utility of computers. The true but unprovable statements concern sweeping properties of logic itself, not the specific kinds of properties we are concerned about in formal methods.[39] Gödel's incompleteness theorems are best seen as affirmations of the remarkably expressive power of arithmetic, rather than limitations on the everyday applicability of logic. The practical limitation on our ability to decide whether or not a certain statement is valid is our ability to find a proof, not whether a proof exists.

There are other arithmetic theories besides Peano's. For example, there are axiomatizations (mostly due to Dedekind [Ded63]) of the rational and the real numbers [Fef89, Sup72]. Axiomatizations of the reals suffer from limitations as surprising as those imposed by the incompleteness theorems. The Löwenheim-Skolem Theorem says that if a first-order theory has a model of infinite cardinality, then it has models of all infinite cardinalities.[40] In particular, this means that any axiomatization of the real numbers has a model that is only countably infinite (i.e., has only as many elements as there are natural numbers). Since Cantor showed (by his "diagonal" argument), that the real numbers are not countably infinite (i.e., they cannot be put into 1-1 correspondence with the natural numbers), this means that no axiomatization of the real numbers can capture the properties of the reals *uniquely*—there will always be *nonstandard* models that satisfy the axioms, but are different from the reals. This discovery is not as disappointing as it may seem: it led to the invention of nonstandard analysis [Rob74], which provides a consistent interpretation to infinitesimals and allows analysis to be built up without recourse to the usual notions of limits and convergence, thereby providing a rigorous reconstruction of early treatments of the calculus [Dau88, Lak78]. Neither do these limitations diminish the practical utility of formal systems of arithmetic in computer science. In fact, nonstandard analysis provides a basis for formal verification of the accuracy of certain floating point calculations [Pra92].

[39] A few unprovable formulas of a genuinely mathematical (as opposed to logical) character are now known: "Since 1931, the year Gödel's Incompleteness Theorems were published, mathematicians have been looking for a strictly mathematical example of an incompleteness in first-order Peano arithmetic, one which is mathematically simple and interesting and does not require the numerical encoding of notions from logic. The first such examples were found in 1977" [PH78]. The example in the cited paper concerns an extension to the Finite Ramsey Theorem.

[40] Infinite cardinalities are explained in Section A.8. There are actually two parts to the Löwenheim-Skolem Theorem as I have stated it: the "downward" half, which allows smaller models to be obtained from bigger ones, is the original Löwenheim-Skolem Theorem; the "upward" half, which allows larger models to be obtained from smaller ones, is really a consequence of the Compactness Theorem. The Compactness Theorem says that if every finite subset of a set of sentences has a model, then the set of sentences has a model.

Since we can formalize Cantor's argument for the uncountability of the reals, there must be some countable model that validates this theorem. A simpler observation of the same kind is "Skolem's Paradox": the set of subsets of the natural numbers must be uncountable (Cantor's Theorem establishes that the powerset of a set has greater cardinality than the original set [Dun91, Chapter 12]), but the formalization of this result is satisfied in some countable model. These are really not the paradoxes they seem: the sets concerned will be uncountable *in the model*, but countable in the "real universe" [EFT84]. That is, the model will be sufficiently impoverished that we cannot construct an injection from the interpretation of the set to that of the naturals, but in the "real universe" we will have more functions available to us and will be able to construct the injection. (Boolos and Jeffrey provide a good discussion of this topic [BJ89, pp. 152–155].)

Formal systems for integer and rational arithmetic similar to one first investigated by Presburger in 1929 are very useful in computer science. Essentially, these are linear arithmetics with addition, subtraction, multiplication, equality, and a "less than" predicate $<$. (By simple constructions, the predicates $>, \leq$, and $\geq$ can be added as well.) By linear is meant the restriction of multiplications to literal constants—so that $3 \times x$ is admitted, but $c \times y$ (where c is a symbolic constant) and $x \times y$ (where x is a variable) are not. A valuable property of these arithmetics is that they are decidable, and therefore conducive to efficient theorem proving [Rab78]. Since arithmetic is ubiquitous, support tools for formal methods that lack effective automation of arithmetic reasoning are extremely tedious and unproductive to use. Classical Presburger arithmetic is a first-order theory (i.e., it permits quantification), but it allows only simple constants and not function symbols. Because function symbols are also ubiquitous (they can arise, through the process of Skolemization, even in formulas that do not include them originally), variations on Presburger arithmetic that make different tradeoffs to maintain decidability can be more useful in practice. In particular, tools for formal methods often use Presburger arithmetic restricted to the propositional (i.e., ground) case only, but with extensions (such as equality with uninterpreted function symbols) [NO79, Sho77, Sho79, Sho84] that are undecidable in the quantified theory. These arithmetics are adequately expressive for most problems in computer science, and formal methods tools incorporating efficient implementations of their decision procedures can be very effective in practical applications.

A.5.3 Simple Datatypes

Peano's axioms serve as a prototype for axiom systems specifying other datatypes commonly used in computer science, such as lists, trees, and so on. Usually, these datatypes have some *constructors*, which are constants or functions that generate

values of the kind considered (e.g., 0 and *succ* in Peano arithmetic), and some *accessors*, which are functions that reverse the process—breaking a value of the kind under consideration into the components that generated it (there are no accessors in Peano arithmetic as I defined it; the *predecessor* function would be an accessor if it were added to the theory). Datatypes require some axioms to specify the relationship among the constructors and accessors, others to specify that values constructed from equal components are equal, and another that specifies an induction scheme.

As an example, here is a theory of *lists*, with constructors Λ (a constant, representing the empty list) and *cons* (a function that builds a new list from a term and an existing list), and accessors *car* and *cdr*.[41] $\mathcal{L}$ is a predicate that recognizes lists; $\mathcal{L}(l)$ is true exactly when l is a list. The first three axioms state that Λ and *cons* construct lists, and that Λ is different from any list constructed using *cons*.

- $\mathcal{L}(\Lambda)$,
- $\mathcal{L}(l) \supset \mathcal{L}(cons(x, l))$,
- $\mathcal{L}(l) \supset cons(x, l) \neq \Lambda$.

The next two axioms describe the relationship between the accessors *car* and *cdr* and the constructor *cons*:

- $\mathcal{L}(l) \supset car(cons(x, l)) = x$,
- $\mathcal{L}(l) \supset cdr(cons(x, l)) = l$,

  Note $car(\Lambda)$ and $cdr(\Lambda)$ are left unspecified; they are usually taken as errors. More sophisticated logics allow them to be explicitly disallowed.

Next is an *extensionality* axiom (one that says values are equal if their components are).

- $\mathcal{L}(l_1) \wedge \mathcal{L}(l_2) \wedge l_1 \neq \Lambda \wedge l_2 \neq \Lambda \wedge car(l_1) = car(l_2) \wedge cdr(l_1) = cdr(l_2) \supset l_1 = l_2$.

The following is an example of what is sometimes called an *eta* rule (one that says a value is equal to that constructed from its components); it is a lemma that can be proved from extensionality.

- $\mathcal{L}(l) \wedge l \neq \Lambda \supset l = cons(car(l), cdr(l))$.

Next, we have a structural induction scheme: to prove a property ϕ of a general list l, it is enough to prove it for Λ and to prove that if it is true for a list l', then it will also be true for the list $cons(x, l')$.

[41] Sometimes the names *hd* (head) and *tl* (tail) are used instead of *car* and *cdr*, respectively.

- $(\phi(\Lambda) \wedge (\forall x, l'\colon \mathcal{L}(l') \wedge \phi(l') \supset \phi(cons(x, l')))) \supset (\forall l\colon \mathcal{L}(l) \supset \phi(l))$.

One consequence of the induction scheme is that all lists are either Λ or a *cons*:

- $\mathcal{L}(l) \supset l = \Lambda \vee \exists x, l' : l = cons(x, l')$.

Finally, we have a *length* function that can be used in termination arguments for recursively-defined functions on lists:

- $length(\Lambda) = 0 \wedge length(cons(x, l)) = length(l) + 1$.

(More generally, we sanction functions defined in a similar manner to that used for *length*.)

Many datatypes commonly used in computer science (such as trees, stacks, etc.) can be specified by axioms similar to those shown above for lists. Because they are so regular, some computer systems supporting formal specification languages can generate suitable axioms automatically for a certain class of data structures, given only a very compact description of the datatype concerned. For example, PVS [ORS92] generates axioms equivalent to all those shown above (they are slightly different because PVS is a typed logic) from the following specification.

```
list[t:TYPE] : DATATYPE
 BEGIN
  null: null?
  cons (car: t, cdr: list): cons?
 END list
```

The "shell" mechanism of the Boyer-Moore prover [BM88] provides similar capability.

A.5.4 Set Theory

During the nineteenth century, several mathematicians attempted to provide a defensible basis for the manipulations performed in analysis (canceling by dx and the like). Cauchy, for example, gave precise definitions to the notions of limit and convergence that had troubled mathematicians since the invention of calculus. Dedekind and Peano's abstract formulations of the real and natural numbers suggested that it might be possible to construct all of mathematics on a few broad, basic principles. Central to these constructions were the ideas of set theory (mainly developed by Cantor) and logic.[42]

[42] Sources for this material include Hatcher [Hat82], Levy [Lev79], Fraenkel [FBHL84], and Shoenfield [Sho78a].

Frege was the first to develop the notion of logic as a formal system in the modern sense. His system was similar to a modern higher-order system,[43] with a nonlogical component that defined a form of set theory, now known as *naive set theory*. The nonlogical part of Frege's system included a two-place predicate written as infix $\in$ and pronounced "is a member of." The intended interpretation is that $x \in y$ expresses the idea that x is a member of the set y. The important point is that as well as being composed of its members, a set can also be regarded as single entity, and can itself be a member of other sets. Furthermore, it is essential to the constructions of Dedekind and Cantor that sets can have infinitely many members.

It seems reasonable to suppose that two sets are equal if and only if they have the same members. This is called the principle of *extensionality*, and it was an axiom of Frege's system. A set can be specified *extensionally* by simply listing all its members: for example

$$y = \{red, blue, green\}.$$

Another way to specify a set is by stating a property that its members must satisfy: for example,

$$y = \{x \mid \phi(x)\}$$

is the usual notation for saying that the set y consists of exactly those members x satisfying the property ϕ. This is called specifying a set *intensionally*. The idea that every property determines a set (i.e., that we are allowed to specify sets intensionally) is the principle of set *comprehension* (also called *abstraction*). The principle of comprehension, which was another axiom of Frege's system, can be expressed as

$$(\exists y : (\forall x : x \in y \equiv \phi(x))).$$

It says that for any property (wff) ϕ of one free variable, there is a set y consisting of exactly those x satisfying ϕ.

Frege showed that he was able to construct the building blocks of mathematics (such as the natural numbers) within his system, and gave convincing arguments that it could serve as a foundation for the whole of mathematics. Unfortunately, this plan was destroyed by Russell in 1902 [Rus67], who pointed out that Frege's system is inconsistent, and therefore unsound (any sentence can be proved in an inconsistent system). *Russell's Paradox*, as it is called,[44] is simply the principle of comprehension with $x \notin x$ for $\phi(x)$. Intuitively, this can be interpreted as follows: if every predicate determines a set, then consider the set y determined by the predicate

[43] Higher-order systems are described in Section A.10.

[44] It should really not be called a paradox; it is a plain contradiction. Other contradictions were known in naive set theory prior to Russell's discovery. These, too, are generally called paradoxes, although technical literature often uses the term *antinomies*. These other antinomies (for example, those of Cantor and Burali-Forti) involve ideas from set theory (in particular, infinite cardinal and ordinal numbers), whereas Russell's goes to very heart of logic itself.

$x \notin x$. That is, y is the set of all sets that are not members of themselves. Now, is y a member of itself or not? If it is, then it satisfies its defining predicate, so that $y \notin y$—that is, it is *not* a member of itself. Conversely, if y is not a member of itself, it does not satisfy the defining predicate, so that $y \in y$—that is, it *is* a member of itself. Either way we obtain a contradiction.

Frege acknowledged that Russell's Paradox destroyed his system's foundation [Fre67] but it was left to others to reconstruct those foundations on a secure footing. The basic problem is in the unrestricted principle of comprehension; fixing the foundations requires placing some control on the way this principle is employed. There are two main approaches by which this can be done. *Axiomatic set theory* is one approach, *type theory* is the other. Here I sketch the axiomatic approach; type theory is described in Section A.10.

The idea behind axiomatic set theory is to allow new sets to be constructed only from existing sets—that way, we avoid things getting out of hand and leading to the antinomies. The best known axiomatic set theory is called Zermelo-Fraenkel (or ZF), after its inventors. In the most austere presentations of ZF, everything is a set (i.e., has zero or more members); sometimes "urelements" (also called "individuals") are provided as well—these can be members of sets but cannot themselves have members. Since everything can be encoded in set theory without urelements, and since it makes some of the statements simpler, I will consider only sets. Also note that ZF uses a very limited predicate calculus, in which $\in$ and $=$ are the only predicate symbols, and there are no function symbols. Functions and additional predicates (not to mention numbers, and the whole of mathematics) are later constructed *within* ZF.

There are eight axioms in ZF; they can all be stated in several different forms (see [FBHL84] for an extended discussion and [Hal84] for an examination of the underlying intuitions), and I will merely describe them, rather than give their formal statements, here.

Extensionality: says that two sets are equal if they have the same members. In symbols

$$(\forall x: x \in a \equiv x \in b) \equiv a = b.$$

We can introduce the notion of subset by a similar construction:

$$(\forall x: x \in a \supset x \in b) \equiv a \subseteq b,$$

but notice that this is merely a definition (i.e., a metalinguistic abbreviation), not an axiom.

Pair: says that if we have two sets a and b, then we can form a new set whose members are just a and b. This set is called the *pair-set* of a and b. By taking

$a = b$, this also allows us to form the set $\{a, a\}$, which by extensionality is the same as the singleton set $\{a\}$. (This axiom can be dispensed with if certain technical adjustments are made to provide "function-classes" [Lev79]).

Separation: says that given a set a, we can form a new set b consisting of just those members of a satisfying a property ϕ. This is similar to the unsound axiom of comprehension, except that members of the set defined by the property are required to come from some existing set a. This construction is usually written

$$b = \{x \in a \mid \phi(x)\}.$$

Union: says that if we have a set a (of sets), then we can form a new set consisting of the members of all its members. This set is sometimes called the *sum-set* and written $\bigcup a$.

Iterated application of the axioms of pairing, separation, and union allow us to construct many familiar sets. For example, we can define the *union* $a \cup b$ of two sets a and b to be the sum-set of their pair-set. Then we can construct the set $\{a, b, c, d\}$ as the union of two pairs: $\{a, b\} \cup \{c, d\}$. Intersection and set difference can be defined using separation:

$$a \cap b = \{x \in a \mid x \in b\},$$

$$a \setminus b = \{x \in a \mid x \notin b\},$$

and the properties of associativity and distributivity can be proved from these definitions.

Power set: says that the powerset of a set is a set (i.e., we can talk of the set of all subsets of a given set). The powerset of a set a is usually denoted $\mathcal{P}(a)$ or 2^a.

Infinity: says that there is an infinite set. The axioms introduced so far tell us how to combine existing sets to yield new ones, but they do give us any sets with which to start the process. If we suppose that we had a set to start with—call it X, say, then we could form the emptyset by separation: $\emptyset = \{x \in X \mid x \neq x\}$. We could then form an infinite collection of sets by a recurrence such as

$$\begin{aligned} n_0 &= \emptyset \\ n_{i+1} &= n_i \cup \{n_i\} \end{aligned}$$

(so that $n_0 = \emptyset, n_1 = \{\emptyset\}, n_2 = \{\emptyset, \{\emptyset\}\}, n_3 = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$ and so on[45]). But although we then have an infinite *collection* of sets, we have nothing that

[45] Note the difference between the emptyset and the set having the emptyset as its only member.

allows us to call that collection a *set*, and no other way to be sure that there is an infinite set.

There are many ways to state the axiom of infinity, but one way is to say that the collection of sets n_i defined as above is a set (it is usual also to state that the emptyset is a set, in order to avoid the need to "seed" the process with some arbitrary set X). The set of n_i created in this way is called ω, and it plays a fundamental role in the development of numbers (the number 2, say, is *defined* to be the set n_2 and so on), but how do we know it is infinite? One definition of an infinite set (due to Dedekind) is one whose members can be put into one-to-one correspondence with some strict subset of itself. Since the members of ω can be put into one-to-one correspondence with the strict subset $\omega \setminus n_0$ (just associate n_i with n_{i+1}) we see that ω is infinite. The particular set ω might seem an arbitrary choice, so more general statements of the axiom of infinity sanction the *kind* of recurrence we used to create the members n_i of ω, without singling out that particular construction.

Replacement: says that if f is a function and x is a set, then the collection of all $f(y)$ for $y \in x$ is a set. I have expressed this axiom in terms of a function f, but functions are not primitive in ZF (I will define them shortly), so formal statements of the axiom can be rather complicated.

To understand why it is needed, consider the infinite collection of sets defined by:

$$\begin{aligned} V_0 &= \emptyset \\ V_{i+1} &= \mathcal{P}(V_i) \end{aligned}$$

(so that $V_0 = \emptyset, V_1 = \{\emptyset\}, V_2 = \{\emptyset, \{\emptyset\}\}, V_3 = \{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \{\emptyset, \{\emptyset\}\}\}$ and so on). We would like to say that this collection is a set, and the axiom of replacement allows us to do so by exhibiting the function f such that $f(n_i) = V_i$. Since the n_i form a set (by the axiom of infinity), so do the V_i.

Most of the axioms of ZF were first stated by Zermelo, but the axiom of replacement is usually credited to Fraenkel (although Skolem and von Neumann have stronger claims). The importance of this axiom is that it is what allows the construction of the "higher" infinities (e.g., by allowing the collection $\{\omega, \mathcal{P}(\omega), \mathcal{P}(\mathcal{P}(\omega)) \ldots\}$ to be a set; we then take its sum-set, and start taking powersets again, and so on).

Regularity (also called **foundation**): says that every nonempty set (of sets) has a member that has no members in common with the original set. This is a technical axiom, intended to eliminate unintuitive possibilities, such as sets which are members of themselves.

Among its consequences are that every set is *well-founded*: that is, if a is any set and $a_0 \in a$, and $a_1 \in a_0$ ...and $a_{i+1} \in a_i$ and ..., then this sequence eventually terminates.[46] Well-foundedness allows us to prove things about sets by induction over the $\in$ relation. Non well-founded sets have been studied recently, and may have some application in computer science [Acz88].

The eight axioms given above are generally augmented by one more:

Choice: says that for any set of sets, there is a *choice function* that selects a member from each of the member sets. (By the axiom of replacement, we can then form a set containing just the members so chosen.) This axiom was added to support the constructions used in a number of important theorems (e.g., Zermelo's well-ordering of the reals).

ZF plus the axiom of choice is generally denoted ZFC and is considered an adequate foundation for the whole of mathematics.

The axiom of choice is rather different from the other axioms because of its essentially nonconstructive character: it asserts that a choice function exists, but doesn't tell us how to construct one. For this reason, the axiom is sometimes considered a little suspect, and there have been many attempts to find more constructive replacements for it. It turns out that weaker axioms are unable to support some parts of mathematics, and alternative axioms of adequate power (e.g., "every set can be well-ordered" or the result known as Zorn's Lemma) are all equivalent to the axiom of choice [RR63].

Gödel showed in 1939 that the axiom of choice was consistent with the other axioms (i.e., if you couldn't get a contradiction from the other axioms, adding the axiom of choice wouldn't enable you to get one). In 1963, Cohen showed that the negation of the axiom of choice is also consistent with the other axioms—so that this axiom is truly independent of the others.

ZFC minus the axiom of replacement is called Zermelo set theory, and is strictly weaker than ZFC, since it excludes certain transfinite sets. There are other set theories, of which the best known is that of von Neumann, Bernays and Gödel (known as NBG or VNB set theory). This theory has a notion of (proper) "classes" as well as sets (proper classes can have sets as members, but cannot themselves be the members of sets or proper classes). NBG is sometimes preferred to ZFC because it has a finite axiomatization.[47]

[46] A relation $\succ$ is well-founded if there are no "infinite descending chains"—i.e., no infinite sequences $a_0 \succ a_1 \succ \ldots \succ a_i \succ \ldots$ A well-founded set is one such that the $\ni$ relation ($a \ni b \equiv b \in a$) is well-founded. This and other properties of relations are described in more detail in Section A.7.

[47] It is also a *conservative extension* of ZF. Conservative extensions are described in the next section.

Some of these distinction can be important when mechanizing formal methods. Although ZF is used by most tools for formal methods based on set theory, Mizar, for example, uses a set theory due to Tarski and Grothendieck [Rud92].

The operator known as Hilbert's ε symbol is related to the axiom of choice. When ϕ is some wff with a single free variable x, then $\varepsilon x : \phi$ denotes "the x such that ϕ"—that is some value that satisfies ϕ, if any such exists, otherwise some arbitrary value. The symbol can be specified by the axiom scheme

$$(\exists x\colon \phi(x)) \supset \phi(\varepsilon x\colon \phi)$$

and can be used to define a choice function (for a set a, let ϕ be $x \in a$). If we denote ZF plus the ε symbol as ZF_ε, then it might look as though ZF_ε = ZFC. This is not so, however, for ZF_ε merely adds the ε operator, it does not strengthen the axioms of separation and replacement to allow wffs involving ε (see [FBHL84, page 73, footnote 1] and [Lei69, section 4.4]); thus, although we could use ε to select representatives from a set of sets, we cannot say that this collection of representatives is a set. Hilbert's Second ε Theorem states that any theorem of ZF_ε not involving ε in its statement is also a theorem of ZF (i.e., ZF_ε is a conservative extension of ZF and so the axiom of choice is not a theorem of ZF_ε). If the axioms of separation and replacement are extended to allow wffs involving ε, then we do obtain the axiom of choice.[48]

I have already sketched how familiar set operators such as $\cup$ and $\cap$ can be defined within ZF, next I briefly describe how relations and functions are constructed. This process is essentially similar to "programming" in a very restricted language.

First, the *ordered* pair (a, b) is represented by the set $\{a, \{a, b\}\}$ (this "coding trick" is due to Kuratowski and Wiener). Then we can define the Cartesian product $x \times y$ of two sets x and y by

$$x \times y = \{(a, b) \mid a \in x \wedge b \in y\}.$$

This construction can be justified (i.e., we can establish the right-hand side is a set) by observing that

$$(a \in x \wedge b \in y) \supset (a, b) \in \mathcal{P}(\mathcal{P}(x \cup y))$$

and then using the separation axiom. A *predicate* P on a set x is a subset of x, and I generally write $P(a)$ for $a \in P$. A *relation* R between x and y is defined to be a subset of $x \times y$ and I usually write aRb for $(a, b) \in R$ (if $x = y$, we say a relation *on* x).

[48] In systems (such as PVS) that are based on higher-order logic (see section A.10), it is a simple exercise in specification and theorem proving to state and prove a type-restricted form of the axiom of choice from the axiom for ε.

In set theory, quantified expressions of the form $(\forall x : x \in S \supset \phi(x))$ are usually abbreviated to $(\forall x \in S : \phi(x))$. Similarly, $(\exists x \in S : \phi(x))$ is an abbreviation for $(\exists x{:}\, x \in S \wedge \phi(x))$. Using these notations, the *domain* of R is the set $\{a \in x \mid \exists b \in y : aRb\}$ (i.e., the set of all first components of pairs in R), and its *range* is the set $\{b \in y \mid \exists a \in x : aRb\}$ (i.e., the set of all second components of pairs in R). (Relations on three or more sets can be defined by iterating the construction). The *domain restriction* of R to a set s is $\{(x,y) \in R \mid x \in s\}$ and is denoted $s \lhd R$; The *range restriction* of R to a set s is $\{(x,y) \in R \mid y \in s\}$ and is denoted $R \rhd s$. The *image* of a set s under the relation R is the range of $s \lhd R$; the *inverse image* is the domain of $R \rhd s$. The *inverse* relation R^{-1} of R is given by $R^{-1} = \{(b,a) \in y \times x \mid (a,b) \in R\}$.

A relation f is a *function* from x to y if $(\forall a \in x{:}\, (\forall b, c \in y : a\, f\, b \wedge a\, f\, c \supset b = c))$ (i.e., if at most one member of the range relates to each member of the domain); f is *total* if its domain is the whole of x (otherwise it is *partial*); it is *surjective* if its range is the whole of y. A function is *injective* if $(\forall c \in y{:}\, (\forall a, b \in x : a\, f\, c \wedge b\, f\, c \supset a = b))$ (i.e., if at most one member of the domain relates to each member of the range); a function is *bijective* if it is both injective and surjective. When f is a total function and $a \in x$, the unique b such that $a\, f\, b$ is denoted $f(a)$. It is a matter of considerable debate how $f(a)$ should be treated when f is partial and $a \in x$ is not in its domain; some of the options are discussed in Section A.11.2.

Although ZF set theory provides the underpinnings for some well-known specification notations such as Z [Spi88], there are rather few theorem proving systems for classical set theory. Ontic [McA89], and the theorem prover (called "Never") of the Eves system [CKM+91], are the most substantial. Also, the generic theorem prover Isabelle [Pau88] has an instantiation for ZF set theory. Decision procedures for fragments of set theory have been extensively studied by Cantone and others (see, e.g., [Can91]). Over a period of 20 years, the Mizar project has formalized and mechanically checked an extensive body of mathematics from a set-theoretic foundation [Rud92]. The Mizar proof-checker seems to provide very little automation, however.

A.6 Definitions and Conservative Extension

In Section A.3 I explained how new propositional connectives could be introduced either as abbreviations (a metalogical approach), or by extending the formal system with additional axioms. In this section I examine how new predicate and function symbols can be introduced into a formal system.[49]

It is clear that we can introduce new predicate or relation symbols as abbreviations (they are then generally called *defined symbols*) and that we do not add

[49] Sources for this section include Gordon [Gor88] and Shoenfield [Sho67].

anything new (other than notational convenience) to the formal system by so doing (since we can always simply expand the abbreviations). Thus, if we already have the relation $<$ available, we can introduce $\leq$ as a defined symbol using

$$x \leq y \stackrel{\text{def}}{=} x < y \vee x = y.$$

If we want $x \leq y$ to be an expression in the object language, then we must expand that language by adding $\leq$ as a new predicate symbol, and then supplying

$$x \leq y \equiv x < y \vee x = y$$

as a new axiom.

Now there are many sentences involving $\leq$ that we might have chosen to add as its "defining axiom," and some of these might be dangerous. For example,

$$(x \leq y \vee x = y) \wedge (x \leq y \supset (x < y \vee x = y))$$

enables us to deduce a contradiction (from $3 \neq 2$, deduce $3 \leq 2$ from the first conjunct, and hence the contradiction $3 < 2 \vee 3 = 2$ from the second). What we need are some simple rules that stop us ruining our formal system by introducing inconsistencies under the guise of defining a new symbol. The requirement that a defining axiom should leave the formal system consistent is a little too strong, since the system might have already been inconsistent; what we can require is that the defining axiom should add no *new* inconsistencies or, equivalently, that if the formal system was consistent before, then it will remain so after the addition of our defining axiom.

A definition is presumably intended to introduce some new concept to the formal system, not to add new properties to existing concepts. Thus we should require that the addition of a definition to a theory creates a *conservative extension* of the existing theory. The enlarged theory is an *extension* of the original if all theorems of the original remain so in the enlarged theory; the extension is *conservative* if every theorem of the enlarged theory that is expressed in the language of the original theory is also a theorem of the original theory. Whether or not an extension is conservative can depend on how the additions are made. For example, if we have a theory $\mathcal{T}$ to which we add two new constants a and b to form a theory $\mathcal{T}'$, then $\mathcal{T}'$ is indeed a conservative extension of $\mathcal{T}$. But if we then add the axiom $a = b$ to create the theory $\mathcal{T}''$, then $\mathcal{T}''$ is not a conservative extension of $\mathcal{T}'$, since $a = b$ is a theorem of $\mathcal{T}''$ that is expressed in the language of $\mathcal{T}'$, but it is not a theorem of $\mathcal{T}'$. However, $\mathcal{T}''$ *is* a conservative extension of $\mathcal{T}$. Distinctions such as this are significant when extensions are made incrementally, since many support systems for

formal methods require that each step is conservative; such systems would allow $\mathcal{T}$ to be extended to $\mathcal{T}''$ in one step, but not in two steps via $\mathcal{T}'$.[50]

Our original search for ways of extending theories without introducing inconsistencies can now be recast as one for rules of definition that guarantee conservative extension. In the case of a predicate symbol P, the rule that the defining axiom should be of the form

$$P(x_1, \ldots, x_n) \equiv \phi,$$

where P does not appear in the wff ϕ and no variables but $x_1, \ldots, x_n$ are free in ϕ does the job. Our original definition for $\leq$ has this form, and is obviously satisfactory.

For a function symbol f, a suitable defining axiom[51] is one of the form

$$y = f(x_1, \ldots, x_n) \equiv \phi,$$

where ϕ is a term not containing f and in which no variables but $x_1, \ldots, x_n$ and y are free, *provided* we can prove the *existence condition*

$$\exists y : \phi$$

and the *uniqueness condition*

$$\phi \wedge \phi[y \leftarrow z] \supset y = z.$$

For example, to introduce the square root function $\sqrt{x}$, the defining axiom would be

$$y = \sqrt{x} \equiv y \times y = x$$

[50] There are more powerful ways of building up consistent theories than by simply making incremental conservative extensions. For example, Robinson's Consistency Theorem says that if two consistent first order systems are in "complete agreement" on their common language (i.e., for each formula ϕ in their common language, at most one of ϕ or $\neg\phi$ is provable in both systems), then the union of the two systems is consistent [BJ89, chapter 24].

[51] It is reasonable to ask whether every concept (i.e., constant, predicate, or function) can be introduced by an explicit defining axiom: might there not be some that can only be defined implicitly by a complex web of axioms? First of all, we can say that an axiom is *independent* of the others in a theory if there is a model for the others that falsifies the one under consideration. (E.g., non-Euclidean geometries are interpretations of Euclid's system that make the Fifth Postulate false but all the others true, thereby demonstrating that the Fifth postulate is independent of the others.) Padoa extended this idea to a method for showing that a concept q is independent of others $p_0, \ldots, p_n$ under axioms $A_0, \ldots, A_m$: if there are two models that both give the same interpretations to $p_0, \ldots, p_n$ and both valuate $A_0, \ldots, A_m$ to true, but give different interpretations to q, then q is independent of the other concepts. Beth's *Definability Theorem* shows that Padoa's technique is complete: if a concept q is, in fact, independent of the others in a theory, then Padoa's technique will succeed in demonstrating that fact. From this it follows that any concept that is implicitly definable in first-order logic is also explicitly definable (see [BJ89, chapter 24] or [Kle67, § 57]).

(so that ϕ is $y \times y = x$), the existence condition would be

$$\exists y\colon y \times y = x,$$

and the uniqueness requirement would be

$$y \times y = x \wedge z \times z = x \supset y = z$$

(which might be hard to prove on the reals!).

An important special case is the one where y does not appear in ϕ; in this case, the defining axiom is equivalent to the equation

$$f(x_1, \ldots, x_n) = \phi$$

and the existence and uniqueness conditions are always provable. This convenient result can be extended to recursive definitions (i.e., those in which f appears in ϕ) provided they have certain simple forms.

The simplest such form is that of *primitive recursion*:

$$\begin{aligned} f(0, x_1, \ldots, x_n) &= g(x_1, \ldots, x_n) \\ f(i+1, x_1, \ldots, x_n) &= h(f(i, x_1, \ldots, x_n), i, x_1, \ldots, x_n) \end{aligned}$$

where g and h are already defined functions that need not take all of the arguments $x_1, \ldots, x_n$ (nor i in the case of h). This can be also be written in the form

$$\begin{aligned} f(i, x_1, \ldots, x_n) = \textbf{if } i = 0 \quad & \textbf{then } g(x_1, \ldots, x_n) \\ & \textbf{else } h(f(i-1, x_1, \ldots, x_n), i-1, x_1, \ldots, x_n). \end{aligned}$$

(Clearly, the induction variable i need not be the first argument.) As an example, observe that Peano's axioms for multiplication

$$\begin{aligned} x \times 0 &= 0 \\ x \times (i+1) &= x \times i + x \end{aligned}$$

are primitive recursive ($n = 1$, $g(x) = 0$, $h(z, i, x) = z + x$) and can also be written in the alternative form

$$x \times i = \textbf{if } i = 0 \textbf{ then } 0 \textbf{ else } x \times (i-1) + x.$$

When we enlarge our formal system with constants, functions, and predicates that define some new "datatype," we often wish to define additional functions by recursion on the structure of the datatype. For example, if we have introduced lists,

with Λ, *cons*, *car*, and *cdr*, we might want to define a function *sum* that adds up the value of all the nodes. We might define this by

$$\begin{aligned} sum(\Lambda) &= 0 \\ sum(cons(x, l)) &= x + sum(l) \end{aligned}$$

or, equivalently, by

$$sum(l) = \textbf{if } l = \Lambda \textbf{ then } 0 \textbf{ else } car(l) + sum(cdr(l)).$$

Plainly, we could prove a metalogical theorem that this construction provides conservative extension just as primitive recursion does, and we could even prove more general results for all datatypes defined by certain structured sets of axioms. However, another way to justify these definitions is by providing a *measure* function from the recursion variable (here l) to the natural numbers and proving that the value of this function strictly decreases across the recursion. In this case, the obvious measure function is the *length* of the lists concerned, and the theorem we would have to prove is

$$l \neq \Lambda \supset length(cdr(l)) < length(l),$$

which will be provable in any well constructed theory of lists.

There are several tricky details that need to be taken care of in specification languages that provide a "definitional principle" for recursions of this sort. For example, taking the identity function as the measure, the definition

$$silly(n) = \textbf{if } n = 0 \textbf{ then } 0 \textbf{ else } silly(n - 2) \times n$$

might appear conservative, despite the fact that it "steps over" the "termination condition" $n = 0$ when applied to an argument that is an odd number. Obviously, the definitional principle must eliminate this possibility if it is to be sound, but methods for doing this are best examined in the context of the particular language concerned.[52]

Not all functions are primitive recursive. The standard example is a function due to Ackermann, which in its modern form is given by

```
ack(m,n) = if m = 0 then n+1
           elsif n = 0
              then ack(m-1, 1)
              else ack(m-1, ack(m, n-1)).
```

[52] In PVS [ORS92], for example, the definition of *silly* would be considered type-incorrect because the recursive argument $n - 2$ cannot be shown to be a natural number (unlike $n - 1$, which could be shown to be a natural number if n is, in the context $n \neq 0$).

This function cannot be defined by (first-order) primitive recursion.[53] However, a definition such as this can be shown to be conservative by demonstrating that the "size" of its pair of arguments (notice that *ack* is recursive in both its arguments) decreases across the recursions according to a *lexicographic* ordering on pairs:[54]

$$(m_1, n_1) < (m_2, n_2) \equiv m_1 < m_2 \vee (m_1 = m_2 \wedge n_1 < n_2).$$

Here, we need

$$\begin{aligned} (m-1, 1) &< (m, 0), \\ (m-1, ack(m, n-1))^{55} &< (m, n), \text{ and} \\ (m, n-1) &< (m, n), \end{aligned}$$

which are all true in the lexicographic ordering.

We might try to justify use of lexicographic ordering to ensure conservative extension using a naive extension to the measure function approach. That is, we could look for a function *size* that would map the pair of arguments to *ack* into a single number that is strictly decreasing across the recursions:

$$\begin{aligned} size(m-1, 1) &< size(m, 0), \\ size(m-1, ack(m, n-1)) &< size(m, n) \text{ and} \\ size(m, n-1) &< size(m, n). \end{aligned}$$

A plausible function is

$$size(m, n) = \xi \times m + n,$$

provided ξ is big enough to ensure

$$\xi \times (m-1) + ack(m, n-1) < \xi \times m + n$$

(from the second case)—that is

$$\xi > ack(m, n-1) - n.$$

Now $ack(m, n-1)$ grows much faster than n, so the right-hand side is unbounded—and this suggests that ξ needs to be infinite!

In fact, this is not so implausible as it might seem, and it serves to motivate introduction of the transfinite ordinals, which are numbers with the properties we require.

[53] Though it can be defined by a higher-order primitive recursion: that is, a definition restricted to the form of primitive recursion, but with functions allowed as arguments [Gor88, pp. 96, 97].

[54] This ordering is called lexicographic because it is the way words are ordered in a dictionary—first on the initial letter, then on the second, and so on.

[55] Since *ack* is the function whose termination we are trying to prove, it should not appear in its own termination argument. A better form of this obligation is $\forall f\colon (m-1, f(m, n-1)) < (m, n)$. The quantification over all *functions* f (of the appropriate signature) is a *higher-order* construction. Since I have not yet introduced higher-order logic (see section A.10), I will continue with the rather suspect construction that uses *ack* itself.

A.7 Ordinal Numbers

Natural numbers can be used in two ways. If we have the members of a set somehow arranged in order, then we can count off its members, and can speak of the "second" member, and the 365th, and so on. Numbers used in this way are called *ordinals*. Alternatively, we can simply ask how many members there are in the set; numbers used in this way are called *cardinals*. For finite sets, the natural numbers serve as both ordinals and cardinals; with infinite sets, however, things get a little more complicated.

The extension of ordinal and cardinal numbers to infinite sets was the work of Cantor.[56] First, though, it is necessary to distinguish different kinds of infinity. Aristotle distinguished the *potential* from the *actual* or *completed* infinite; the potential infinite typically arises when some variable can take on values without limit. Any particular value is finite, but the range of possibilities is unbounded. Prior to Cantor, it was generally assumed that mathematics could only be concerned with the potential infinite; the actual infinite was considered unfathomable. Cantor broke through this restriction of thought and divided the actual infinite into the increasable infinite, or *transfinite*, and the *absolute* infinite. For Cantor, the transfinite

> "is in itself constant, and larger than any finite, but nevertheless unrestricted, increasable, and in this respect thus unbounded. Such an infinite is in its way just as capable of being grasped by our restricted understanding as is the finite in its way." [Hal84, page 14]

Ordinal numbers are used to count the members of a set arranged in some order, so first we need some terminology for ordering relations. A binary relation $<$ on a set a is said to be

reflexive: if $x < x$,

irreflexive: if $\neg(x < x)$,

symmetric: if $x < y \supset y < x$,

asymmetric: if $x < y \supset \neg(y < x)$,

antisymmetric: if $x < y \wedge y < x \supset x = y$,

transitive: if $x < y \wedge y < z \supset x < z$,

[56] Sources for this section include Cantor [Can55], Hallett [Hal84], Hatcher [Hat82], Johnstone [Joh87] and Phillips [Phi92]. A less technical but very readable introduction to Cantor's work is provided by Dunham [Dun91, chapters 11 and 12].

connected: if $x < y \vee y < x$,

trichotomous: if $x < y \vee y < x \vee x = y$, and

well-founded: if $\forall b \subseteq a : b \neq \emptyset \supset \exists x \in b : \neg\exists y \in b : y < x$ (i.e., every nonempty subset of a has a $<$-minimal member or, equivalently, there are no infinite $<$-descending chains).

Furthermore $<$ is said to be a

preorder: if it is reflexive and transitive,

partial order: if it is reflexive, transitive, and antisymmetric,

total order: if it is reflexive, transitive, antisymmetric, and trichotomous,

strict order: if it is irreflexive, transitive, and antisymmetric (actually, the third is implied by the first two),

strict total (or **linear**) **order**: if it is irreflexive, transitive, antisymmetric, and trichotomous (again, the third is implied by the first two),

well-order: if it is irreflexive, transitive, antisymmetric, trichotomous, and well-founded (actually, the last condition implies the first, the last two imply the second, and the first two imply the third—so that all we really need is well-founded and trichotomous). Equivalently, we can say that a well-ordering is a well-founded linear order.

The idea of the ordinals is that they should be sets with a canonical well-order. Then, since every set has a well-ordering in ZFC (the existence of a well-ordering is equivalent to the axiom of choice), the members of any set can be placed in order alongside those of an ordinal, whose members then act as numbers for all the positions in the line. (The unique ordinal isomorphic to a given set a, well-ordered by $<$, is called the *order-type* of $(a, <)$.)

In ZFC, the only primitive relations are $\in$ and $=$, and of these only $\in$ is a candidate for forming a well-ordering. Then, in order to achieve trichotomy, we will need to construct sets whose members are such that of any distinct pair of members, one is a member of the other (this is *connectedness* on $\in$). It turns out that sets with these properties can be found among the *transitive* sets, where a set a is called transitive if

$$y \in x \wedge x \in a \supset y \in a.$$

(Transitive sets a have the properties $\bigcup a \subseteq a$ and $a \subseteq \mathcal{P}(a)$—i.e., every element of a is also a subset of a.) We then define the *ordinals* as the transitive sets that are well-ordered by $\in$ (or, equivalently, the sets that are both transitive and connected).

$\emptyset$ is an ordinal, and if x is an ordinal, then so is $x \cup \{x\}$ (which I will denote x'). Consequently, the sets $n_0, n_1, \ldots n_i, \ldots$, which were defined in just this way in the previous section but one, are ordinals (they are called the *von Neumann natural numbers*). An ordinal y such that $y = x'$ for some ordinal x is called a *successor*; all other ordinals (except $\emptyset$) are called *limits*. Now, we know (from the axiom of infinity) that the set ω of all the von Neumann natural numbers exists; it can be seen to be transitive and connected, and is therefore an ordinal. It can be shown not to be a successor, and is therefore a limit ordinal (in fact the smallest one). It is also our first infinite number. We can create larger infinite ordinals by iterating the process used to create ω. That is, we can form the successors $\omega', \omega'', \omega''', \ldots$ and then collect these (together with $0, 1, 2, \ldots, \omega$) up into a set, which we will call $\omega \times 2$ and which is another ordinal. In this way we can form $\omega \times 3, \omega \times 4, \ldots$ To get further, we need to define the operations of addition, multiplication, and exponentiation on ordinals. These are each defined by recursion, with separate cases according to whether the second argument is 0, a successor, or a limit.

$$\begin{array}{lcl} \alpha + 0 & = & \alpha \\ \alpha + \beta' & = & (\alpha + \beta)' \\ \alpha + \beta & = & \bigcup\{\alpha + \gamma \mid \gamma < \beta\} \text{ where } \beta \text{ is a limit} \end{array}$$

(The less-than relation $<$ on the ordinals is just set-membership $\in$.) If α is the order-type of a set a, and β the order-type of a set b, then $\alpha + \beta$ is the order-type of the disjoint union of a and b.

$$\begin{array}{lcl} \alpha \times 0 & = & 0 \\ \alpha \times \beta' & = & \alpha \times \beta + \alpha \\ \alpha \times \beta & = & \bigcup\{\alpha \times \gamma \mid \gamma < \beta\} \text{ where } \beta \text{ is a limit} \end{array}$$

If α is the order-type of a set a, and β the order-type of a set b, then $\alpha \times \beta$ is the order-type of the Cartesian product $a \times b$ under lexicographic ordering.

$$\begin{array}{lcl} \alpha^0 & = & 1 \\ \alpha^{\beta'} & = & \alpha^\beta \times \alpha \\ \alpha^\beta & = & \bigcup\{\alpha^\gamma \mid 0 < \gamma < \beta\} \text{ where } \beta \text{ is a limit} \end{array}$$

If α is the order-type of a set a, and β the order-type of a set b, then β^α is the order-type of the function space $a \rightarrow b$.

Note that the first two cases in each set of equations are the same as the corresponding Peano axioms (with the $'$ operator in place of *succ*), so that the successor ordinals (and, in particular the von Neumann natural numbers) behave just like the natural numbers. The case of the limit ordinals is a little different, however, for

addition and multiplication are not commutative in these cases (e.g., $1 + \omega = \omega$ whereas $\omega + 1 = \omega'$, and $2 \times \omega = \omega$ whereas $\omega \times 2 = \omega + \omega$).

In this way we can form ordinals up to ϵ_0, which is the limit of the sequence $\omega, \omega^\omega, \ldots, \omega^{\omega^{\omega^{\cdot^{\cdot^{\cdot}}}}}$ and has the property $\omega^{\epsilon_0} = \epsilon_0$. (The ordinals continue beyond this, but for formal methods ϵ_0 is usually adequate.)

Each nonzero ordinal α below ϵ_0 has a unique representation (called Cantor normal form) with the structure

$$\alpha = \omega^{\alpha_0} \times a_0 + \omega^{\alpha_1} \times a_1 + \cdots \omega^{\alpha_n} \times a_n,$$

where $\alpha \geq \alpha_0 \geq \alpha_1 \geq \cdots \geq \alpha_n$ are ordinals, and $a_0, a_1, \ldots a_n$ are nonzero natural numbers.

Constructive representations of the Cantor normal form are quite convenient in formal methods for establishing "termination" arguments for recursions on tree-like data structures, and are used, for example, in the Boyer-Moore prover and in PVS. (The ordering relation $<$ can be defined by primitive recursion on the ordinals below ϵ_0, so its own well-definedness is easily established in most formal methods). Returning to the example of Ackermann's function that motivated this introduction of the ordinals, we can now see that our treatment of its termination becomes satisfactory if we set $\xi = \omega$.

The definitions given above for ordinal addition, multiplication, and exponentiation are examples of definition by *transfinite recursion*. There is a corresponding induction scheme called *transfinite induction*, which can be described as follows:

$$\text{If} \left[\begin{array}{l} \phi(0) \wedge \\ \forall \alpha \in \mathcal{O} : \phi(\alpha) \supset \phi(\alpha') \wedge \\ \forall \beta \in \mathcal{O}_L : (\forall \gamma < \beta : \phi(\gamma)) \supset \phi(\beta) \end{array} \right] \text{ then } \forall \alpha \in \mathcal{O} : \phi(\alpha).$$

Where $\alpha \in \mathcal{O}$ means that α is an ordinal, and $\alpha \in \mathcal{O}_L$ that it is a limit ordinal.

Another way to write transfinite induction is

$$(\forall \alpha \in \mathcal{O} : (\forall \beta \in \mathcal{O} : \beta < \alpha \supset \phi(\beta)) \supset \phi(\alpha)) \supset (\forall \gamma \in \mathcal{O} : \phi(\gamma)).$$

Written in this form, it can be seen that transfinite induction owes more to the fact that $<$ is a well-ordering, than to the fact that the values concerned are ordinals. This gives rise to *well-ordered* induction, which is simply the second form of transfinite induction generalized to the case where $\mathcal{S}$ is a set well-ordered by $<$

$$(\forall \alpha \in \mathcal{S} : (\forall \beta \in \mathcal{S} : \beta < \alpha \supset \phi(\beta)) \supset \phi(\alpha)) \supset (\forall \gamma \in \mathcal{S} : \phi(\gamma)).$$

In fact, this induction scheme is sound when $<$ is merely a well-founded relation, in which case it is known as *well-founded induction* (it is also called *Noetherian*

induction, after Emmy Noether). Well-founded induction is the most general induction scheme: transfinite induction, and all the other induction schemes that we have seen, can be derived from it. It is a great convenience if tools for formal methods are able to support these very general induction schemes.

A.8 Cardinal Numbers

The transfinite ordinals are needed to establish the soundness of certain constructions in formal methods. Transfinite cardinal numbers, on the other hand, find little application in formal methods, so I will just briefly mention them.[57]

The *cardinality* (or cardinal number) of a set a, often written $|a|$, is the "number" of members it contains. This concept of "number" is straightforward for finite sets, but needs care when we consider infinite sets. The key idea (due to Cantor) is to start by defining two sets to have the same "size" (the technical term is *equipollent*) if the members of one can be put into bijective (i.e., 1-1 and onto) correspondence with those of the other. A set a is smaller than a set b if there is an injection (i.e., 1-1 function) from a to b, but not vice-versa.[58]

The cardinal number of a set could be defined as some canonical member chosen from the collection of sets that have the same size. The trouble is that this collection is too big to be a set, so we cannot define it within set theory. However the ordinals having the same size as a given set do constitute a set; and since any set of ordinals is well-ordered, we can choose the least member of that set to be the canonical representative that is the cardinal number of the original set. It is easy to see that the transfinite cardinals are chosen from the limit ordinals. The smallest transfinite cardinal is ω; when considered as a cardinal, it is usually written $\aleph_0$ and pronounced "aleph-null." Sets of cardinality $\aleph_0$ (i.e., those whose members can be put into bijective correspondence with the natural numbers) are called *countably* or *denumerably* infinite. (Recall Dedekind's definition that a set is infinite if it has the same size as some strict subset of itself.) All the ordinals mentioned in the previous section, including ϵ_0 are countable. The smallest uncountable ordinal is denoted Ω; as a cardinal it is denoted $\aleph_1$.

Cantor's theorem demonstrates that the size of any set is strictly less than that of its powerset,[59] so we know $|\omega| < |\mathcal{P}(\omega)|$. We have defined $|\omega| = \aleph_0$, and cardinal arithmetic is defined so that $|\mathcal{P}(\omega)| = 2^{\aleph_0}$. Thus $\aleph_0 < \aleph_1 \leq 2^{\aleph_0}$. The

[57] This section draws on Halmos [Hal60].

[58] The Schröder Bernstein theorem states that if there is an injection from a to b, and vice-versa, then the two sets have the same size. Its proof is a standard test piece in mechanized theorem proving. (It is the transfinite cases that are interesting.)

[59] This is another standard small problem for mechanized theorem proving. Again, it is the transfinite cases that are interesting.

continuum hypothesis conjectures that the second of these relations is not strict: i.e., $\aleph_1 = 2^{\aleph_0}$. Now the set of real numbers has the same size as $\mathcal{P}(\omega)$, so another (in fact, the original) way to state the continuum hypothesis is that the cardinality of the reals is the least uncountable cardinal. (The set of real numbers is also known as the continuum, hence the name of the hypothesis.) The continuum hypothesis is known (by results of Gödel and Cohen) to be independent of the axioms of set theory.

A.9 Many-Sorted Logic

Classical first-order theories treat all individuals as belonging to some single universe. It is sometimes convenient to distinguish different *sorts* of individuals within the universe. For example, we may want some variables to represent natural numbers, others reals, and so on. Many-sorted logic allows first-order systems to be extended in this way. Mathematicians generally use subscripts to indicate the sort to which a variable belongs, so that x_σ, for example, indicates a variable x of some sort called σ. Quantification in many-sorted logic extends only over the sort of the variable concerned, so that

$$(\forall x_\sigma : \phi(x))$$

means that the formula ϕ with free variable x is true whenever x is replaced by any value of sort σ (for simplicity, I generally drop the sort subscript once the variable has been introduced). Functions and predicates also indicate the sorts of values over which they operate. For example, the function *abs* whose value is the natural number that is the absolute value of its integer argument will be described as having *signature* $\mathcal{Z} \rightarrow \mathcal{N}$, where $\mathcal{Z}$ is the sort of integers, and $\mathcal{N}$ that of natural numbers. As with individuals, the signature of a function can be indicated in formulas as a subscript, for example

$$x_\mathcal{Z} \neq 0 \supset abs_{\mathcal{Z}\rightarrow\mathcal{N}}(x_\mathcal{Z}) > 0.$$

The model theory of predicate calculus is adjusted in the many-sorted case so that each sort is interpreted by its own specific "carrier set" and various technical adjustments are made (for example, sorts should generally not be empty) so that the resulting system is sound and complete. Many-sorted systems are very natural for computer science, since computer programs usually need to distinguish the different kinds of objects represented and manipulated.

A.10 Typed Systems and Higher-Order Logic

While Zermelo and Fraenkel developed axiomatic set theory, Russell [WR27] explored a different approach to the construction of a consistent set theory.[60] Considered from one perspective, Russell's paradox demonstrates that Frege's axiom of comprehension is dangerously unrestricted. This is, essentially, the perspective of axiomatic set theory: the ZF axioms break the unrestricted connection between predicates and sets given by the axiom of comprehension and substitute more constrained ways of constructing sets. But there must be more to it than this, for it is possible to reproduce Russell's paradox without ever mentioning sets: simply consider the predicate that characterizes those predicates that do not exemplify themselves.[61] Predicates such as this are *higher order*: their construction involves predicates that apply to predicates, and quantification over predicates. For example, if we abbreviate the predicate that characterizes those predicates that do not exemplify themselves by $PNET$ (for Predicates Not Exemplifying Themselves), then the definition for this predicate is $PNET(P) \equiv \neg P(P)$. Russell's paradox is obtained by substituting $PNET$ for the free predicate variable P in this definition to obtain $PNET(PNET) \equiv \neg PNET(PNET)$.

Frege's system was higher-order, and we can now see that axiomatic set theory does much more than replace the axiom of comprehension with the axioms of ZF: it also eliminates all functions and all predicates but $\in$ and $=$ from the logic and restricts quantification to the first-order case (i.e., variables can range only over sets).[62] ZF then reconstructs functions and predicates *within* set theory (as sets of pairs) and in this way develops an adequate foundation for mathematics. This approach is that associated with the *formalist* school of Hilbert, and is essentially mathematical and pragmatic; it is mathematical because it is based on essentially mathematical intuitions. No one would claim that the axiom of replacement, say, is an elementary truth, fundamental to all rational thought; rather, it is a mathematical construction (and a pragmatically motivated one at that).

Unlike the formalists, Frege and Russell were *logicists*: they wanted logic to comprise *exactly* the fundamental truths of rational thought, and then wanted to show that mathematics followed from such a logic: they didn't want to axiomatize mathematics, they wanted to *derive* it. What Russell sought to do, therefore, was

[60] Sources for this material include Andrews [And86], Hatcher [Hat82], Benacerraf and Putnam [BP83], van Bentham and Doets [vBD83], and Hazen [Haz83].

[61] A predicate exemplifies itself if it has the property that it characterizes: for example, the predicate that characterizes the property of being a predicate is clearly a predicate, and does exemplify itself; whereas the predicate that characterizes the property of being a man is not a man, and so does not exemplify itself.

[62] The idea that set theory should be based on first-order logic was due to Skolem. A very readable account of the emergence of first-order logic is given by Moore [Moo88].

to retain all the generalities of Frege's system, with arbitrary function and predicate symbols, and higher-order quantification (because these are necessary to capture informal logical discourse, and to derive mathematics without additional axioms), but to find some minimal and plausibly motivated restriction that would keep the antinomies at bay.

Russell and Poincaré, in the course of an exchange of letters, decided that the source of the antinomies was what Russell called "vicious circle" constructions, and what Poincaré called *impredicative* definitions [Gol88]. An impredicative definition is one that has a particular kind of circularity; it is not the circularity of recursion, but one where a thing is defined by means of a quantification whose range includes the thing being defined. Thus, a set b is defined impredicatively if it is given by forms such as

$$b = \{x \mid \forall y \in a\colon P(x, y)\},$$

where P is a predicate and b may be an element of a.

In work culminating in the Principia Mathematica [WR27] (written with A. N. Whitehead), Russell developed his *ramified theory of types* that augments higher-order logic with rules that exclude impredicative definitions.[63] There are two components to the ramified theory: *types* and *orders*. The second of these causes certain technical difficulties, and Russell introduced an "axiom of reducibility" to get around them. Chwistek and Ramsey thought that this axiom vitiated the purpose of orders, and that one might as well have started out with just the notion of types and never bothered with orders. However, this simpler theory is not predicative and is vulnerable to some paradoxes that are avoided by the ramified theory.

Accordingly, Ramsey (building on an observation of Peano) divided the paradoxes into those he called *logical* (such as Russell's, Cantor's and Burali-Forti's), and those he called *semantic* or *epistemological* (such as the Liar, Richard's, Grelling's, and Berry's[64]); he then argued that the epistemological paradoxes are the concern of philosophy or linguistics, not logic, since they concern the interpretation of concepts like "nameable," not basic problems in the machinery of deduction. Ramsey argued that the requirement on a logic is that it should exclude the logical paradoxes. He

[63] Limitation to predicative definitions is a powerful restriction: it eliminates construction of the Dedekind cut, and with it a good part of analysis, and also Cantor's theorem and the general theory of cardinal numbers and the higher infinities. Furthermore, from the Platonist perspective adopted by most mathematicians (namely, that mathematical objects really exist), there is nothing objectionable in impredicative definitions: the "vicious circle" disappears if we conceive of the offending definitions as selecting some member from a pre-existing collection. Consequently, few mathematicians espouse the rigors of a predicative type theory. (See Hazen [Haz83] for a readable account of predicative logics.)

[64] Berry's is the easiest of these to describe: consider "the least natural number not nameable by an English phrase of less than 200 letters." The material in quotation marks is an English phrase of less than 200 letters that names this number.

noted that types are adequate, on their own, for this; orders and the complexities of the ramified theory are needed only to exclude impredicative definitions and the epistemological paradoxes. Ramsey [Ram90] then reconstructed Russell's system (principally by eliminating orders) so that "its blemishes may be avoided but its excellences retained" to yield what is now called the *simple theory of types* (its modern formulation is due to Church).[65]

Roughly speaking, simple type theory avoids the logical paradoxes of naive set theory by "stratifying" sets according to their "type" or "level." Individual elements can be considered to belong to level 0, sets of those elements to level 1, sets of sets of elements to level 2, and so on. The comprehension axiom

$$(\exists y : (\forall x : x \in y \equiv \phi(x)))$$

is then restricted to the case where x is of a lower level than y. I say "roughly speaking" because simple type theory is a logic of *predicates* rather than sets: instead of saying "x is a member of the set y" (i.e., $x \in y$), in type theory it is more common to say "x has property y" (i.e., $y(x)$). In terms of predicates, simple type theory requires that each predicate has a higher type than the objects to which it applies (and therefore a predicate cannot apply to itself). The "definition" of the predicate *PNET* given earlier is inadmissible because its right-hand side is not well-typed.

I just said that simple type theory is a logic of predicates, but it is usually built on a more primitive foundation that takes (total) functions as the basic elements, with predicates as a special case. Within this framework, functions are allowed to take other functions as arguments and may return functions as values. The *type* of a function indicates the "level" at which it operates in a more sophisticated way than the simple numbering scheme for levels suggested earlier. Thus, if ι is the type of individuals (which can be considered as functions of zero arguments), the type of functions from individuals to individuals can be denoted $\iota \to \iota$, and a function on those functions will have type $(\iota \to \iota) \to (\iota \to \iota)$.[66]

It is generally convenient to use a many-sorted foundation, and to subsume sorts within the type structure, so that a function type $(\mathcal{Z} \to \mathcal{N}) \to \mathcal{N}$ indicates

[65] The simple theory of types is not the same as the ramified theory plus the axiom of reducibility: the simple theory is *extensional*, whereas the other is not. Extensional means that coextensive properties are considered identical (i.e., $(\forall x{:}\, \phi(x) = \psi(x)) \supset \phi = \psi$). Examples of nonextensional properties are "propositional attitudes": a person may *want* or *wonder about* one proposition but not another even though they are, in fact coextensive. The classic example is:

Scott is the author of Waverley,
King George wonders whether Scott is the author of Waverley,
King George wonders whether Scott is Scott,

whose absurdity is taken as evidence that "wonders whether" in nonextensional. Unlike the simple theory, the ramified theory of types (even with the axiom of reducibility) can give an account of nonextensional properties [Haz83, section 5].

[66] Types allow us to distinguish $(\iota \to \iota) \to (\iota \to \iota)$ from $(\iota \to \iota) \to \iota$, which cannot be done by simply numbering "levels."

a function whose values are of type[67] natural number and whose arguments are functions from integers to natural numbers. Within this framework, we distinguish a type $\mathcal{B}$ (for "Boolean," though mathematicians generally write it o), and then view predicates as functions returning type $\mathcal{B}$. Thus, the predicate $<$ on the natural numbers can be considered a function of type $\mathcal{N} \times \mathcal{N} \rightarrow \mathcal{B}$. Next, since the principle of comprehension holds that every predicate determines a set, we dispense with the separate notion of set and simply identify sets with predicates: $x \in y$ is identified with $y(x)$.[68]

In this scheme, dropping some of the more tedious subscripts, the comprehension axiom becomes

$$(\exists y_{\sigma \rightarrow \mathcal{B}} : (\forall x_\sigma : y(x) \equiv A_\mathcal{B}))$$

where $A_\mathcal{B}$ is a formula in which y does not occur free. The typing restrictions attached to x and y prevent the kind of self-reference that leads to the paradoxes of naive set theory. The same construction can be used to assert the existence of general functions as well as predicates:

$$(\exists y_{\sigma \rightarrow \tau} : (\forall x_\sigma : y(x) = A_\tau)). \tag{A.2}$$

The function $y_{\sigma \rightarrow \tau}$ whose existence is thereby asserted can be denoted

$$(\lambda x_\sigma : A_\tau)_{\sigma \rightarrow \tau}.$$

The λ here is called the *abstraction* operator; $(\lambda x_\sigma : A_\tau)_{\sigma \rightarrow \tau}$ is the name of the function whose value on argument x_σ is A_τ, where x_σ presumably occurs free in A_τ. For example,

$$(\lambda x_\mathcal{Z} : (\textbf{if } x < 0 \textbf{ then } -x \textbf{ else } x)_\mathcal{N})_{\mathcal{Z} \rightarrow \mathcal{N}}$$

is the "absolute value" function on the integers. λ is a variable-binding operator, just like the quantifiers.[69]

Using this notation, the comprehension axiom (A.2) becomes

$$(\forall x_\sigma : (\lambda x_\sigma : A_\tau)_{\sigma \rightarrow \tau}(x_\sigma) = A_\tau)$$

[67] Actually this is a sort, but the distinction between sorts and types is generally dropped in type theory.

[68] Note that the restrictions of sorts and types mean that sets are homogenous in simple type theory: it is not possible to form sets such as $\{x, \{x\}\}$, since x and $\{x\}$ are of essentially different types.

[69] In fact it is possible to *define* the quantifiers in terms of λ by means of the construction

$$(\forall x : \phi) \stackrel{\text{def}}{=} (\lambda x\colon \phi) = (\lambda x\colon true).$$

and since the innermost x_σ is bound by the λ, whereas the others are bound by the $\forall$, we can instantiate the $\forall$-bound instances with a term t_σ to obtain

$$(\lambda x_\sigma : A_\tau)_{\sigma\to\tau}(t_\sigma) = A_\tau[x_\sigma \leftarrow t_\sigma],$$

which is the rule of *β-conversion*.

Church first developed a system for defining and manipulating functions defined by λ-abstraction. His system is called the λ-calculus, and it provides a foundation for the semantics of functional programming languages. In addition to β-conversion, there are two other "conversions" (sometimes also called "reductions") in the λ-calculus.

alpha-conversion: this is the renaming of bound variables—that is,

$$(\lambda x_\sigma : A_\tau(x))_{\sigma\to\tau} = (\lambda y_\sigma : A_\tau(y))_{\sigma\to\tau}.$$

eta reduction: this says that a function equals its own abstraction—that is,

$$(\lambda x_\sigma : f_{\sigma\to\tau}(x))_{\sigma\to\tau} = f_{\sigma\to\tau}.$$

In axiomatic set theory, individuals, sets, sets of sets, and so on all belong to the same universe, and quantification extends over the entire universe. In type theory, the universe is stratified by the type hierarchy, so that quantification applies to each type separately. This means that type theory allows (in fact, requires) quantification over functions and predicates. For this reason, it is also known as *higher-order logic*: propositional calculus allows no quantification and can be regarded as "zero-order" logic, predicate calculus allows quantification over individuals and is also known as "first-order" logic, "second-order" logic allows quantification over functions and predicates of individuals, "third-order" allows quantification over functions and predicates of functions, and so on up to ω-order logic, which allows quantification over arbitrary types. In this scheme, type theory or higher-order logic corresponds to ω-order logic.


In higher-order logic, functions can take other functions as arguments and return functions as values (functions that do so are sometimes called *functionals*, and the term *function* is then reserved for those functions that operate on, and return, simple values). Functions of several arguments can be treated as functionals of a single argument in higher-order logic by a process known as "Currying" (named after the logician H. B. Curry, although it was actually first described by Schönfinkel [Sch67]). To see how this works, suppose we have a function f that takes two arguments and returns the result of adding the first to the square of the second:

$$f \stackrel{\text{def}}{=} (\lambda x_\mathcal{N}, y_\mathcal{N} : x + y^2)_{\mathcal{N}\times\mathcal{N}\to\mathcal{N}}$$

so that we have $f(2,3) = 11$. We could instead define a functional f' that takes a single argument and returns a *function* that takes another single argument and returns the square of that value, plus the argument to the first functional:

$$f' \stackrel{\text{def}}{=} (\lambda x_{\mathcal{N}} : (\lambda y_{\mathcal{N}} : x + y^2)_{\mathcal{N}\to\mathcal{N}})_{\mathcal{N}\to(\mathcal{N}\to\mathcal{N})}.$$

In this case we have $f'(2) = (\lambda y : 2 + y^2)_{\mathcal{N}\to\mathcal{N}}$, so that $f'(2)(3) = 11$. In general, a function application $g(x, y, \ldots, z)$ is equivalent to a sequence of Curried applications $g'(x)(y)\cdots(z)$. Currying makes it easier to define the logic since it is only necessary to consider functions and predicates of one argument (I am implicitly doing this in the presentation here). Some formulations and mechanizations of higher-order logic (e.g., HOL) *require* all functions to be Curried, whereas others merely allow it. The mandatory use of Curried functions makes for unfamiliar constructions and is a difficult hurdle for many users; however, the freedom to use functionals is very valuable when used appropriately.

Type theory or higher-order logic has the same axioms and rules of inference as first-order logic with equality (generalized to allow quantification over functions and predicates), together with the type-restricted form of the comprehension axiom given earlier, and an axiom of *extensionality*:

Extensionality: $(\forall x_\sigma : f_{\sigma\to\tau}(x) = g_{\sigma\to\tau}(x)) \supset f = g$.

This simply says that two functions (or predicates) are equal if they take the same value as each other for every value of their arguments.


There are two main classes of models for type theory. The details are very technical but a rough idea can be conveyed as follows. Since we can quantify over functions in type theory, we have to ask what we mean when we speak of *all* functions from D to D, say. Under the *standard* interpretation, we really do mean all functions on the domain that interprets D. A formula is valid if it valuates to true in all such interpretations. On the other hand, we might want to interpret *all* as ranging over just some particular class of functions (e.g., the continuous functions). In this latter case, the interpretation is relative to what is variously called a *frame* [Mon76] or a *type structure* [And86] that identifies the particular class of functions concerned. Under the *general models* interpretation of type theory (due to Henkin), a formula is valid if it valuates to true in *all* frames or type structures. Since the standard interpretation is included as one frame among those of the general models interpretation, it is clear that *fewer* formulas are valid under the general models interpretation than under the standard one. The standard interpretation is the more natural one, but the axioms given above are not complete in this case (though they are sound and consistent). The advantage of the general models

interpretation is that the axiomatization is complete for this case (it cuts out the valid but unprovable formulas), and several other metamathematical properties can also be established. Furthermore, under the general models interpretation, there is a way to translate or interpret higher-order logic formulas into (sorted) first-order logic.

The logicist's hope was that with type theory as a "self evident" foundation, it would be possible to derive mathematics in a consistent way without the need to introduce additional axioms such as those of Peano, or Zermelo-Fraenkel:

> "What were formerly taken, tacitly or explicitly, as axioms are either unnecessary or demonstrable" [WR27, preface].

Natural numbers can be defined using an idea due to Frege: 3, for example, is the predicate that designates the property of being a triple. I will not give the formal definitions here, since they are tricky, but the idea is that 0 is the property of being the empty set, and the property of being the successor $succ(n)$ to some natural number n is that of being a set that can be formed by adding an element to a set that has the property n.[70] The problem with this approach is that there is nothing that requires the individuals to be infinite in number, and so we cannot guarantee that any large numbers exist. This difficulty is overcome by adding an *axiom of infinity* to the other axioms of type theory. There are several ways to state this axiom, but one way is to assert that there exists some relation $<$ on the individuals that is irreflexive, transitive, and unbounded (i.e., $(\forall x : (\exists y{:}\, x < y))$). With this foundation, it is possible to derive Peano's axioms as theorems.

As noted earlier, type theory without the axiom of infinity is consistent and (under the general models interpretation) complete. Since the addition of the axiom of infinity allows us to develop arithmetic, it is not surprising, in light of Gödel's results, that the full system (with the axiom of infinity) is incomplete. In terms of power, this form of simple type theory is equivalent to Zermelo set theory (i.e., ZF without the axiom of replacement). For computer science, the consequent loss of the higher infinities is unimportant.

Type theory is notationally complex[71] and has some properties that mathematicians find inconvenient. For example, the empty set is not a simple notion: for

[70] Since sets and properties are equivalent in type theory, another way to say this is than n is the set of all n-element sets. This way of looking at things led to much criticism: surely the set of all triples, conceived as a totality, is "an extravagant affair" [Car58, page 141]. This indicates the significant difference between set theory and type theory. In set theory, sets are conceived as totalities, and to avoid the paradoxes we have to make sure they do not get "too big." In type theory, sets are conceived as determined by their properties, and to avoid the paradoxes we must obey the restrictions of typing.

[71] Especially using the mathematicians' notation; they Curry all functions, drop all punctuation, reverse the order of the type-symbols, and write function application without parentheses—see Andrews [And86], for example.

each type, the empty set of elements of that type is distinct from the empty set of a different element type. This duplication seems perfectly acceptable on linguistic grounds (e.g., is having no money the same as having no worries?); rather less so is the realization that the construction of the natural numbers undertaken above was performed relative to a type (I glossed over this), so that every type has its own version of the natural numbers, and its own arithmetic.[72]

Type theory also lacks some of the interesting metamathematical properties of first-order logic, such as the Compactness, Löwenheim-Skolem, and Interpolation Theorems.[73] This is no accident, for some of the metamathematical properties of first-order logic are unique: Lindstrom's Characterization Theorem says that first-order logic is the "strongest" logic with both a compactness theorem and a (downward) Löwenheim-Skolem Theorem. Since these two theorems are the principal tools for deriving metamathematical results in first-order logic, logicians are reluctant to forsake them, and many therefore regard first-order logic as the only "true logic." Barwise [Bar85] observes that this attitude confuses the subject matter of logic with one of its tools. He likens first-order logic, a tool constructed to help investigate the general conception "logic," to the telescope, a tool constructed to help study the universe. Logicians who assert that logic *is* first-order logic are like those who would claim that astronomy is the study of telescopes.

Although first-order logic has many interesting properties, there are several useful and important notions that cannot be defined within it; for example, none of *infinite* set, *countable* set, *open* set, *continuous* function, *well-ordering*, and *random* variable are first-order definable. These can all be defined in type theory, however. Other concepts, such as induction or Hilbert's ε operator, which require set-theoretic constructions or metalogical assistance from schemes to specify in first-order logic, can also be stated simply and directly in type theory.

For formal methods in computer science, the expressive power of type theory may be more important than the metamathematical richness of first-order logic. Moreover, the complex notation of type theory can be considerably streamlined when used as a specification language for formal methods: computer scientists are used to typed systems in programming languages, and their practice of "declaring" variables and constants before use supplies type information implicitly, without the sub- or superscripts of the mathematicians notation; "overloading" (allowing a single symbol such as + or "emptyset" to denote several similar functions of

[72] Fraenkel, Bar-Hillel, and Levy call the "reduplication" of notions such as numbers and the empty set "repugnant" [FBHL84, page 160].

[73] The first two of these were described earlier; the last, often called Craig's Interpolation Lemma (or Theorem), says that if $\phi(A, B) \supset \psi(B, C)$, where $\phi(A, B)$ denotes a formula built up from names in collections A and B, then there is a formula $\gamma(B)$ (i.e., one built up from such names as are common to both ϕ and ψ) such that $\phi(A, B) \supset \gamma(B) \supset \psi(B, C)$. The Interpolation Theorem is used in the proof of Beth's and Robinson's Theorems.

different types that are distinguished by context) also contributes to simple, familiar notation.[74] The duplication of numbers and arithmetics can be overcome by simply adopting Peano's axioms, so that the (canonical) natural numbers are available as a base sort.[75] The incompleteness of type theory under the standard interpretation renders it unattractive to metamathematicians, but this is of little concern in formal methods, since we will need arithmetic and other incomplete theories in any case.

The great advantages of type theory as a foundation for formal specification languages are the very strong and mechanized "typechecking" that can be provided (this makes for very effective early error-detection in specifications), and the expressive convenience of higher-order constructions and quantification. The fact that everything is built on total functions also allows for rather effective mechanical theorem proving in type theory (recall that in set theory, total functions are a special case).

Formal methods and theorem proving systems based on the simple theory of types include HOL [GM93], PVS [ORS92], and TPS [AINP88].

A.11 Additional Topics

In this section I briefly touch on a number of topics. The aim here is simply to give readers exposure to certain terms they may run across in the literature or in discussions on formal methods. My treatment will be very brief and rather superficial, but should at least point to the relevant literature and help the reader identify the field to which a particular concept or piece of terminology belongs.

A.11.1 Modal Logics

Aristotle and, later, medieval logicians conceived of many *modes* of truth, such as *necessary* truths, and those things that *ought* to be true, or that we *know* to be true, or *believe* to be so.

It is possible to formalize these notions by adding the *modal operators* (also called *modalities*) $\Box$ and $\Diamond$ to the propositional calculus (thereby yielding a propositional modal logic).[76] For example, $\Box A$ can read as "it is necessary that A," in which case $\Diamond A$ is read as "it is possible that A."

[74] Russell employed a similar approach, which he called "typical ambiguity": type symbols were usually omitted, and the user was expected to mentally supply them in a manner that provided a well-formed construction.

[75] This would be repugnant to Russell, since he wanted to minimize use of axioms; but for formal methods we are concerned with utility, not philosophical purity.

[76] Mints [Min92] is a good introduction to modal logic.

All modal logics define $\Diamond$ in terms of $\Box$ by the identity

$$\Diamond A \stackrel{\text{def}}{=} \neg\Box\neg A.$$

They also include all the theorems of propositional calculus (generalized to include those with modal operators, such as $\Box A \supset \Box A$), modus ponens, and the inference rule of *necessitation*:

$$\frac{A}{\Box A}.$$

What axiom schemes are added depends on the readings ascribed to the modal operators. There are several axiom schemes with standardized names:

K: $\Box(A \supset B) \supset (\Box A \supset \Box B)$
D: $\Box A \supset \Diamond A$
T: $\Box A \supset A$
4: $\Box A \supset \Box\Box A$
B: $A \supset \Box\Diamond A$
5: $\Diamond A \supset \Box\Diamond A$.

Modal logics are given names such as KT4, which identifies the logic that contains just the axiom schemes K, T, and 4. Some standard modal logics are KD, KT4 (also known as S4), KT5 (also known as S5), and KD45. S5 is historically important as the logic of necessity, but more recently it has become widely used in computer science as a logic of knowledge ($\Box A$ is then read as "it is known that A"). Using this modal logic to reason about "who knows what" has proved a powerful way to analyze distributed systems [HF89]. S4 gives rise to many temporal logics ($\Box A$ is then read as "always A" and $\Diamond A$ as "sometimes" or "eventually" A), which can be used to reason about the properties of concurrent and distributed systems. The modal logic KD45 can be used as a logic of belief ($\Box A$ is then read as "it is believed that A") and has found applications in reasoning about cryptographic key distribution protocols [Ran88]. KD is a "deontic" logic: one for reasoning about obligations.

Leibniz was the first to conceive a recognizably modern semantics for modal logic. He imagined that there could be many worlds: necessary truths would be those that are true in all worlds, possible truths would be those that are true in only some of them. Models for modal logics are based on this idea of a "possible world" semantics (also called "Kripke" semantics after Saul Kripke, who developed the formal treatment while still in his teens). The details are a little too lengthy to go into here, but a key component is an "accessibility" relation between possible worlds; the various standard modal logics can then be characterized by the algebraic properties of the accessibility relation in their models. For example, S5 is characterized by an accessibility relation that is an equivalence relation. These topics are developed in standard texts on modal logics, such as those by Hughes and Cresswell [HC68, HC84], and Chellas [Che80].

As noted, *temporal* logics are modal logics (generally specializations of S4) in which the operators $\Box$ and $\Diamond$ are read as "always" and "eventually," respectively. Pnueli [Pnu77] was the first to recognize that these logics could be used to reason about distributed computations and, in particular, about liveness properties (that is, about things that must happen eventually). There are two families of temporal logics: linear time and branching time [Lam83]. Both families have their adherents, and both have led to effective specification techniques. It is usually necessary to embellish the basic logics of either family with additional operators in order to achieve a comfortable degree of expressiveness; examples include "next state," "until," and backwards-time operators. Interval logics are temporal logics specialized for reasoning over intervals of activity. The Temporal Logic of Actions (TLA) [Lam91] is a temporal logic in which the modal operators are generalized from states to pairs of states (actions); it achieves considerable expressiveness with very little mechanism.

As with predicate calculus, a valid formula in modal logic is one that valuates to true in all interpretations. Many modal logics have what is called the "finite model property," which renders them decidable. The models of temporal logic are essentially finite-state machines; conversely a finite-state machine is a potential model for a temporal logic formula. This observation gives rise to "model checking": the goal is to check whether a finite-state machine describing a system implementation satisfies a desired property specified as a temporal logic formula. This process is equivalent to testing whether the specified machine is a model for the specified formula. Because temporal logic can be quite expressive, and because model-checking is decidable, this technique offers a completely automatic means for verifying certain properties of certain systems [CES86]. Very clever model-checking algorithms allow finite-state machines with large numbers of states to be checked in reasonable time [BCM+90]. Model checking is an example of a state-exploration method (recall Section 2.2.3); it is not a replacement for conventional theorem proving in support of verification (it is applicable to only certain properties and implementations), but it can be a very valuable adjunct.

Despite their name, temporal logics do not provide ways to reason about "time" in a direct or quantitative (i.e., "real-time") sense: they provide a tool for reasoning about the order (i.e., temporal sequencing) of events, and about eventuality properties. Several extensions have been proposed to both temporal [AH89, Koy90] and classical [JM86, Ost90] logics for reasoning about real-time properties, but these are best regarded as promising research, rather than techniques ready for immediate exploitation.

All the modal logics I have considered so far are propositional. It is possible to combine modal operators with quantification to yield first-order modal logics but the details prove rather tricky, as exemplified by the following formula.

$$\Diamond \exists x \colon \phi(x) \supset \exists x \colon \Diamond \phi(x).$$

This is called the Barcan formula (after Ruth Barcan), and it is provable if S5 is combined with first-order logic. The difficulty posed by this formula is that, interpreting $\Diamond$ as "possibly," it seems to require that we admit the existence of (or, as logicians and philosophers say, "include in our ontology") merely possible objects. For example, if it is possible that a cow jumped over the moon, then the Barcan formula tells us that there is a specific cow that has possibly performed this act. This ontological commitment is considered objectionable (or, at least, controversial) and so there have been many attempts to combine modal and first-order logics in ways that exclude the Barcan formula (see [Gar84, HC68]).

A.11.2 Logics for Partial Functions

Partial functions are those whose values are not defined for all arguments in their domain: for example, division is not defined when the divisor is zero. The question then is what meaning to ascribe to expressions such as

$$\frac{x}{y} \times y = x \tag{A.3}$$

when it is possible that y could be zero. I briefly mention five main approaches. Cheng and Jones [CJ90] consider a few others but do not, in my opinion, do adequate justice to the third and fifth of the alternatives below. Farmer [Far90] gives a very readable discussion at the beginning of an otherwise very technical paper.

Russell's Theory of Descriptions. In set theory, functions are sets of pairs, and the meaning of $f(x)$ is "the y such that $(x, y) \in f$." The problem is then that of ascribing a meaning to expressions of the form "the such and such" when there is no object with the property "such and such." ("The present king of France" is the standard example.) Expressions of the form "the such and such" are called *definite descriptions* and are intended to single out some unique individual satisfying the description "such and such." How to interpret a definite description when it does not identify a unique individual (either because there is none, or more than one, having the given property) was one to which Russell devoted a great deal of effort in the early 1900s. He used the notation $\iota x\colon \phi(x)$ for "the x such that $\phi(x)$," where ι is the definite description operator.[77] Russell's solution to the problem of how to deal with $\iota x\colon \phi(x)$ when this did not identify a unique individual was a contextual one: he gave no meaning to $\iota x\colon \phi(x)$ considered in isolation, but set the expression in which it occurred to false. Thus, "the present king of France" has no meaning on its own, but "the present king of France is bald" is false in Russell's treatment.

[77] Russell actually used an upside-down ι, but that is a typographical trick that is not so easy to perform in LaTeX as it is with metal type.

Using Russell's operator, the value of $f(x)$ is given by $\iota y\colon (x, y) \in f$; this is well-defined if $\exists! y\colon (x, y) \in f$, otherwise any expression in which it appears is false. The expression (A.3) is therefore false in this treatment when $y = 0$.

One problem with Russell's approach is that it raises the question of scope: how large an enclosing expression should be set to false when a definite description does not denote a unique individual? Discussion of this question occupies much of Russell's papers on the subject (one written with Whitehead is the most accessible [WR67]). Another problem is that identity is no longer reflexive: $f(x) = f(x)$ is false if x is not in the domain of f. For these reasons, Russell's approach is seldom employed today, but his theory of descriptions has influenced much of the work that followed.

All functions are total. In this scheme, every function has some value assigned for each member of its domain—so that $\frac{x}{0}$, for example, does have a value. We can choose whether or not to supply axioms that enable any properties of such "artificial" values to be deduced. If, for example, we choose not to specify any properties of the value $\frac{x}{0}$, then we will not be able to prove (A.3) as a general theorem, though we will be able to prove the weaker form

$$y \neq 0 \supset \frac{x}{y} \times y = x. \tag{A.4}$$

This approach is quite natural in type theory, where (total) functions are primitive. In set theory, it is equivalent to defining the value of $f(x)$ to be $\varepsilon y\colon (x, y) \in f$, where ε is Hilbert's choice operator (recall subsection A.5.4 on page 266).

The arguments against this approach are that it does not correspond to informal mathematical practice (rather than a way to treat partial functions, it is really a way of doing without them), and that we have to be very careful to avoid inadvertently defining (and hence having to implement) properties of the artificial values or, worse, introducing inconsistencies. For example, if we took (A.3) as an axiom, then the case $y = 0$ gives $\frac{x}{0} \times 0 = x$. But another rule of arithmetic is $z \times 0 = 0$, so that the substitution $z \leftarrow \frac{x}{0}$ gives $\frac{x}{0} \times 0 = 0$, which contradicts the earlier result (when x is nonzero).

Functions are total, on a precisely specified domain. This scheme is the same as the previous one, except that it is performed in the context of a logic with a very rich type system, including what are called *predicate* subtypes and *dependent* types, that allow the domains of functions to be specified very precisely.

As its name suggests, a predicate subtype is one associated with a predicate. In the case of division, for example, we can associate the subtype $\mathcal{Q}'$ of the

rationals $\mathcal{Q}$ with the predicate $(\lambda q : q \neq 0)$ and can then type the division operation as a total function $\mathcal{Q} \times \mathcal{Q}' \rightarrow \mathcal{Q}$. An expression like $\frac{x}{0}$ then has no meaning since it does not satisfy the rules of typing. The expression (A.3) is perfectly valid (and can be taken as an axiom), provided y is of type $\mathcal{Q}'$. More interestingly, the theorem (A.4) is valid, even when y is of type $\mathcal{Q}$ (and could, therefore, be zero), because the value of the expression is independent of the type-incorrect application $\frac{x}{0}$: if $y = 0$, then (A.4) is true because the antecedent to the implication is false (and the value of $\frac{x}{0}$ is irrelevant); otherwise y is of type $\mathcal{Q}'$ and (A.3) applies.

Dependent types are most easily understood by means of an example. Consider the function

$$f(x, y)_{\mathcal{R}\times\mathcal{R}\rightarrow\mathcal{R}} \stackrel{\text{def}}{=} \sqrt{x - y}.$$

This function is undefined (on the reals) if its second argument is larger than its first. But if we change the domain of the function from $\mathcal{R} \times \mathcal{R}$ to

$$x{:}\,\mathcal{R} \times \{y{:}\,\mathcal{R} \mid x \geq y\} \tag{A.5}$$

(i.e., to pairs of real numbers in which the first is not less than the second), then the partiality disappears: the function is total on this more precisely specified domain. Types such as A.5 are called *dependent* because the type of the second component depends on the *value* of the first. Dependent types, and several other innovations in the treatment of types, had their origins in the pioneering Automath project of de Bruijn [dB80].

The advantages of this approach over the previous one are that it obviates any need to construct artificial values or axioms for expressions such as $\frac{x}{0}$ or $pop(empty)$, and it disallows (and typechecking will flag) unsafe expressions such as (A.3) when y is of type $\mathcal{Q}$, thereby forcing us, early on, to recognize the need for the guarded form (A.4). Its disadvantage is that typechecking becomes undecidable: theorem proving may be needed to decide the acceptability of formulas. Pragmatically, however, this approach appears quite effective (especially when its mechanization includes a powerful theorem prover), and seems attractively close to informal mathematical practice. Approaches of this kind are used in the Nuprl [CAB+86], PVS [ORS92], and Veritas [HDL89] systems.

Multiple-Valued Logics. If we admit truly partial functions into the logic, then several further choices must be faced. One choice, typified by the "free" logics discussed in the next item, allows terms that have no denotation, but does not allow "undefined" as a value. The alternative introduces a special "undefined" value, often denoted $\perp$ that can be manipulated like an ordinary value.

Once "undefined" is admitted as a value, we have to further decide whether undefinedness is allowed to propagate from the term to the (logical) expression level: for example, does an expression like $f(x) < g(y)$ always yield an ordinary truth value, even if $f(x)$ could be undefined, or could the expression itself be undefined? Attempts to restrict "undefined" to the term level (Russell's treatment attempts to do this) have not been very successful, so it is generally necessary to admit "undefined" into the logic as an additional truth value. The Logic of Partial Functions (LPF) [BCJ84] is of this kind; it provides the logical foundation for the VDM specification notation. A disadvantage of this approach is that all the logical connectives must be extended to the case of undefined expressions and, more importantly, certain properties of traditional logic must be adjusted to retain soundness. In the case of LPF this leads, for example, to the loss of the law of the excluded middle and the deduction theorem.

The main argument against LPF is that it is awkward, nonstandard, and unfamiliar. A consideration of some alternative partial logics, and the identification of one that is free of many of the disadvantages of LPF, has appeared recently [Owe93].

Free Logics. The original motivation for free logics was to avoid certain "paradoxes" that arise when constants are assumed always to refer to objects in the domain of quantification [Ben85]. For example, starting from the identity $K = K$, where K is some constant, we can obtain the theorem $(\exists x : x = K)$ by existential generalization. If K abbreviates "The King of France," then we have just proved that there is a King of France! Free logics avoid this anomaly by introducing a predicate E called the existence predicate ($E(a)$ is interpreted as "a exists") and restricting quantifiers to range only over existing objects. The quantifier rules are modified so that we can deduce $(\exists x : x = K)$ from $K = K$ only if we have already established $E(K)$—which is tantamount to what we are trying to prove.

Logics to deal with partial functions have been proposed along these lines [TvD88, volume I, chapter 2, section 2]. The challenge is to construct a logic that is as faithful to informal mathematical practice as possible. Scott's formulation [Sco79], for example, has the disadvantage that $(\forall x : \phi(x))$ is no longer equivalent to $\phi(x)$. Beeson introduced an alternative system that overcomes most objections [Bee86] (more accessible references are [Bee85, Chapter 6, Section 1] and [Bee88, Section 5]).

Whereas Scott's is a logic of partial *existence* (it allows models with objects that do not satisfy the existence predicate), Beeson's is a logic of partial terms (LPT) that is concerned with the *definedness* of names and terms. LPT in-

troduces the atomic formula $a\downarrow$, where a is a term, with the interpretation "a is defined." The quantifier axioms become

A4′. $(\forall x : \phi(x)) \wedge t\downarrow \supset \phi[x \leftarrow t]$,

A5′. $\phi[x \leftarrow t] \wedge t\downarrow \supset (\exists x : \phi(x))$.

In addition, we have $c\downarrow$ and $x\downarrow$ for each constant symbol c and variable x. Atomic formulas evaluate to true only if their arguments are defined:

$$\phi(s, t, \ldots, u) \supset s\downarrow \wedge t\downarrow \wedge \cdots \wedge u\downarrow, \tag{A.6}$$

where $s, t, \ldots, u$ are metavariables representing terms. Two notions of equality are used: $=$ is strict equality (false if either of its arguments is undefined), and $\simeq$ is "Kleene equality," which is true if both its arguments are undefined:

$$s \simeq t \stackrel{\text{def}}{=} (s\downarrow \vee t\downarrow) \supset s = t.$$

The equality axioms are

$$x = x \wedge (x = y \supset y = x),$$

$$s \simeq t \wedge \phi(s) \supset \phi(t), \text{ and}$$

$$s = t \supset s\downarrow \wedge t\downarrow .$$

(Here x and y are variables). The last of these is a special case of (A.6), as is

$$f(s, t, \ldots, u)\downarrow \supset s\downarrow \wedge t\downarrow \wedge \cdots \wedge u\downarrow$$

(i.e., functions are defined only if their arguments are).

In the case of the example (A.3), we will presumably have $\frac{x}{y}\downarrow \equiv y \neq 0$, so that $(\frac{x}{y} \times y)\downarrow \equiv y \neq 0$. Thus, the left hand side of (A.3) is defined (and equal to x) provided $x \neq 0$, and the right hand side is always defined (i.e., $x\downarrow$). An equation is false if either of its arguments is undefined, so (A.3) is always defined, and is true if $y \neq 0$, and false otherwise. The form (A.4) is unconditionally true.

PX [HN88] is a computational logic based on these ideas, while a (higher-order) logic of this kind is mechanized in the IMPS system [Far90, FGT93], and variants have been proposed for other specification languages [MR91]. Gumb [Gum89, Chapter 5] uses a free logic to express facts about execution-time errors in programs, and Parnas presents a "logic for software engineering" [Par93] that uses similar ideas.

All these approaches to partial functions, except the second, generally require the user to prove subsidiary theorems in order to discharge definedness obligations. The third approach requires these subsidiary proofs to be performed during typechecking, the first, fourth and fifth require them during proof construction. Doing the subsidiary proofs at typecheck time avoids some duplication of effort (the other approaches can require the same subsidiary results to be proved separately in proofs of different theorems), and allows some faults to be detected earlier. In addition to their treatment of partial functions, the rich type systems of the third approach allow terms to be typed more precisely, and thereby increase the information present in a specification [HD92].

A.11.3 Constructive Systems

The earliest conceptions of mathematical existence were constructive: to Euclid, proving that through every point there exists a line parallel to another given line meant that there was a *procedure* for constructing that line (with ruler and compass). There were some early proofs by contradiction (for example, of the irrationality of $\sqrt{2}$), but these dealt with *non*existence or impossibility, rather than existence. More troubling are proofs of *existence* by contradiction. For example, there exist two irrational numbers x and y such that x^y is rational: if $\sqrt{2}^{\sqrt{2}}$ is rational, we are done; otherwise take this number as x and let y be $\sqrt{2}$. This is surely a little unsatisfactory, as we do not know which of the two cases is true.

During the 19th and early 20th centuries, the advent and rapid application of set theory led to many nonconstructive proofs of this kind.[78] One of the most controversial was Zermelo's nonconstructive 1905 proof of the well ordering of the reals. Several mathematicians were dissatisfied with such proofs, but it was Brouwer who launched a full scale assault on the admissibility of nonconstructive methods:

> "the only possible foundation of mathematics must be sought...under the obligation carefully to watch which constructions intuition allows and which not...any other attempt at such a foundation is doomed to failure" [from Brouwer's PhD Thesis, quoted in [Bee85, page 430]].

Brouwer's position became known as *intuitionism*, and although he opposed several nonconstructive techniques and concepts (for example, the completed infinite) it was his criticism of the "law" of the excluded middle [Bro67] (i.e., $\phi \vee \neg\phi$, as used about x^y above) that became the *sine qua non* of intuitionistic mathematics.

[78] The Historical Appendix to Beeson's book [Bee85] is a good, brief source for those who would like to read more of the history of these topics. Beeson's book and the two-volume work of Troelstra and van Dalen [TvD88] are standard texts on constructive mathematics.

Although Brouwer was not particularly interested in creating a formal system for intuitionism (after all, he rejected the axiomatic method as a foundation), Kolmorgorov, Heyting and others developed intuitionistic propositional and predicate calculi that do without the law of the excluded middle; in this context, "ordinary" logical systems that retain the excluded middle are called *classical* logics.

Both Brouwer and the opposing camp (the formalists, led by Hilbert) thought that strict adherence to intuitionistic principles would force the abandonment of large tracts of mathematics (by logicians at least; "practical" mathematics would proceed undisturbed by such arcane disputes). The intuitionists seemed to accept this, but the formalists saw it as a challenge to the relevance and utility of foundational studies (for example, Ramsey writes of "the Bolshevik menace of Brouwer and Weyl" [Ram90, page 229]) and were inspired to shore up their foundations: Hilbert declared

> "Wherever there is any hope of salvage, we will carefully investigate fruitful definitions and deductive methods. We will nurse them, strengthen them, and make them useful. No one shall drive us out of the paradise which Cantor has created for us" [Hil83].

However, Gödel later showed (via what is called the double-negation interpretation) that intuitionistic and classical arithmetic have the same strength [NS93, Section V.6],[79] and Bishop provided a practical demonstration that much of analysis could be developed in an entirely constructive (though not specifically intuitionistic) manner [BB85]. Thus, restriction to constructive methods does *not* seem to limit to an unacceptable degree the amount of mathematics that actually can be done.

What does all this have to do with computers? Well, anything that can be done by a computer is necessarily constructive; thus, if we seek a logical foundation for computing (as opposed to general mathematics) it seems natural to seek it within a constructive framework. Intuitionistic logic is a particularly strong candidate because of what is known as the "Curry-Howard isomorphism," which establishes a direct correspondence between intuitionistic proofs and functional programs:[80] an intuitionistic existence proof can be converted into a program that constructs the thing concerned. An idea of the flavor of this approach can be obtained as follows. In intuitionism, a proof that A implies B means a procedure which will take any (intuitionistic) proof of A and convert it into a proof of B; thus, if we have a program that constructs A, the proof of $A \supset B$ gives us a program that constructs B.

[79] Gödel also demonstrated a connection between modal and intuitionistic logics [KK62, page 680]. ([KK62] is a standard, and accessible, text on the history and development of logic.)

[80] This really goes back to Kleene, who showed that any formula that could be defined in intuitionistic (Heyting) arithmetic was Turing computable; thus, Heyting arithmetic could serve as a high-level programming language.

As with classical systems, intuitionistic logics that are adequate for actually specifying and reasoning about interesting computations require more than just the intuitionistic fragment of predicate calculus: we need arithmetic, sets, and/or types. Constructive type theories, especially those built on ideas first widely canvassed by Martin-Löf, dominate this enterprise (often under the slogan "propositions as types"). Martin-Löf's theories are *not* theories in ordinary predicate calculus, so they cannot be described by simply listing their axioms: the entire apparatus of the system has to be developed, and I do not have space to attempt this here. Interested readers can find a good introduction in [Tho91].

Constructive type theories are the foundation for a number of interesting systems for formal specifications and proofs, of which the oldest and best known is Nuprl [CAB+86] (which is based on, and extends, Martin-Löf's approach), and one of the most recent is COQ [DFH+91] (which mechanizes the "Calculus of Constructions," which itself is derived from a system variously known as the "second-order" or "polymorphic" λ-calculus and as "System-F.") Arguments in favor of these approaches are that they bring programming and proof into intimate correspondence. Arguments against them are that their foundation is unfamiliar to most users, and that the need to constantly discharge existence obligations is tedious. However, very similar proof obligations arise in classical logics with rich type systems when trying to establish admissibility of definitional forms or consistency of axiomatizations. Hence, it may be that the actual practice of formal methods in constructive and classical settings is more similar than might at first appear.

A.11.4 Programming Logics

Functional programs can be specified quite directly in any logic that supports recursive definitions; reasoning about such programs can then be performed directly within the logic concerned. Imperative programs, on the other hand, operate on an implicit "state" that has no counterpart in ordinary logic. For example, the Fortran assignment statement `x = x+1` must be interpreted as saying that the value of `x` in the "new state" is to be equal to `1` plus the value of `x` in the "old state." *Programming logics* are logics that provide ways to specify and reason about imperative programs and their effects on a program state.

Hoare [Hoa69] introduced the notation $\{P\}S\{Q\}$ (often called a *Hoare sentence*) for specifying the behavior of a program fragment S. In this construction, P and Q are expressions involving the "variables" of the program S and the intuitive interpretation is that if execution of S starts in a state in which the expression P is true, and if execution terminates, then it will do so in a state in which Q is true. A typical programming language logic will include the following axiom for the *skip* (do nothing) statement

$$\{P\}\ \textbf{skip}\ \{P\}$$

and the following for the assignment statement

$$\{P[v \leftarrow e]\}\ \texttt{v := e}\ \{P\},$$

where `v` is a "program variable" and `e` is an arithmetic expression, together with axioms or rules of inference for each of the other constructs in the programming language concerned. Standard rules include the following ones for sequential composition, and for the *conditional* and *while* statements, respectively:

$$\frac{\{P\}S_1\{Q\} \wedge \{Q\}S_2\{R\}}{\{P\}S_1; S_2\{R\}},$$

$$\frac{\{P \wedge B\}S_1\{Q\} \wedge \{P \wedge \neg B\}S_2\{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{endif}\ \{Q\}},$$

$$\frac{\{P \wedge B\}S\{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \{P \wedge \neg B\}}.$$

In addition, there will be some rules concerning the interaction between the programming logic and the classical logic that underlies it, such as the following rule of *precondition strengthening*:

$$\frac{(P' \supset P) \wedge \{P\}S\{Q\}}{\{P'\}S\{Q\}}.$$

As an example, let us prove that the statement

if `x<0` **then** `x := -x` **else** **skip** **endif**

causes the final value of x to be the absolute value of its initial value. We can specify this requirement by

$$\{x = X\}\ \textbf{if}\ \texttt{x<0}\ \textbf{then}\ \texttt{x := -x}\ \textbf{else}\ \textbf{skip}\ \textbf{endif}\ \{x = |X|\},$$

where $X$is a logical variable (implicitly universally quantified over the whole Hoare sentence). By the rule for conditional statements, we need to prove

$$\{x = X \wedge x < 0\}\ \texttt{x := -x}\ \{x = |X|\}$$

and

$$\{x = X \wedge \neg x < 0\}\ \textbf{skip}\ \{x = |X|\}.$$

By precondition strengthening, the second of these follows from

$$(x = X \wedge \neg x < 0 \supset x = |X|) \wedge \{x = |X|\}\ \textbf{skip}\ \{x = |X|\},$$

whose first conjuct follows by predicate calculus, arithmetic, and the definition of the absolute value function, and whose second conjuct is an instance of the *skip* rule given earlier. The other case of the conditional follows by precondition strengthening from

$$(x = X \wedge x < 0 \supset -x = |X|) \wedge \{-x = |X|\}\ \texttt{x := -x}\ \{x = |X|\},$$

whose first conjuct follows by predicate calculus, arithmetic, and the definition of the absolute value function as before, and whose second conjuct is an instance of the *assignment* rule given earlier.

A programming logic can be cast into the more familiar form of a classical logic by making explicit the program state that is implicit in the axioms and rules of the programming logic. In this interpretation, the precondition P and postcondition Q of a Hoare sentence $\{P\}S\{Q\}$ are regarded as classical predicates over the "before" and "after" states s and t, respectively, and the program fragment S is regarded as a shorthand for its denotation or "meaning" $[\![S]\!]$, which is a relation on states: $[\![S]\!](s,t)$ is true if execution of the program fragment S, when started in state s, can terminate in state t. Then the interpretation of the Hoare sentence $\{P\}S\{Q\}$ is

$$\forall s,t : P(s) \wedge [\![S]\!](s,t) \supset Q(t).$$

In this interpretation, the rule of precondition strengthening becomes

$$\frac{(\forall s : P'(s) \supset P(s)) \wedge (\forall s,t : P(s) \wedge [\![S]\!](s,t) \supset Q(t))}{(\forall s,t : P'(s) \wedge [\![S]\!](s,t) \supset Q(t))}$$

which is classically valid.

Similarly, the rule for sequential composition becomes:

$$\frac{(\forall s,t : P(s) \wedge [\![S_1]\!](s,t) \supset Q(t)) \wedge (\forall s,t : Q(s) \wedge [\![S_2]\!](s,t) \supset R(t))}{(\forall s,t : P(s) \wedge [\![S_1; S_2]\!](s,t) \supset R(t))}.$$

I have not gone into enough detail, nor developed enough notation, to give the interpretation of the axiom for assignment in the general case (see Gordon [Gor89]), but the general idea can be conveyed by considering the particular example

$$\{-x = |X|\}\ \texttt{x := -x}\ \{x = |X|\}$$

encountered earlier. The interpretation of an integer program "variable" such as x is a function from state to integer (so that $x(s)$ is its value in state s), and the interpretation of this Hoare sentence is then

$$(\forall s,t,X : -x(s) = X \wedge [\![\texttt{x := -x}]\!](s,t) \supset x(t) = |X|).$$

The End

That concludes this rapid introduction to those aspects of mathematical logic that I consider most relevant to formal methods. Experts will notice that I have simplified or ignored some details—but not, I hope, to the point of inaccuracy. I recommend those who wish to read further to consult initially the books noted near the beginning of the section of this appendix that deals with the topic of interest to them.

Bibliography

[Acz88] Peter Aczel. *Non-Well-Founded Sets*, volume 14 of *CSLI Lecture Notes*. Center for the Study of Language and Information, Stanford, CA, 1988.

[AH89] R. Alur and T. A. Henzinger. A really temporal logic. In *30th IEEE Symposium on Foundations of Computer Science*, pages 164–169, 1989.

[AINP88] Peter B. Andrews, Sunil Issar, Daniel Nesmith, and Frank Pfenning. The TPS theorem proving system. In Lusk and Overbeek [LO88], pages 760–761.

[AK88] William Aspray and Philip Kitcher, editors. *History and Philosophy of Modern Mathematics*, volume XI of *Minnesota Studies in the Philosophy of Science*. University of Minnesota Press, Minneapolis, MN, 1988. Outgrowth of a conference held in 1985.

[Ake78] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, June 1978.

[And86] Peter B. Andrews. *An Introduction to Logic and Type Theory: To Truth through Proof*. Academic Press, New York, NY, 1986.

[Bar78a] Jon Barwise, editor. *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, Holland, 1978.

[Bar78b] Jon Barwise. An introduction to first-order logic. In *Handbook of Mathematical Logic* [Bar78a], chapter A1, pages 5–46.

[Bar85] J. Barwise. Model-theoretic logics: Background and aims. In J. Barwise and S. Feferman, editors, *Model-Theoretic Logics*, Perspectives in Mathematical Logic, chapter 1, pages 3–23. Springer-Verlag, New York, NY, 1985.

[BB85] E. Bishop and D. Bridges. *Constructive Analysis*. Springer-Verlag, Berlin, Germany, 1985.

[BCJ84] H. Barringer, J. H. Cheng, and C. B. Jones. A logic covering undefinedness in program proofs. *Acta Informatica*, 21:251–269, 1984.

[BCM+90] J. R. Burch, E. M. Clarke, K. L McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *5th Annual IEEE Symposium on Logic in Computer Science*, pages 428–439, Philadelphia, PA, June 1990. IEEE Computer Society.

[Bee85] Michael J. Beeson. *Foundations of Constructive Mathematics*. Ergebnisse der Mathematik und ihrer Grenzgebiete; 3. Folge · Band 6. Springer-Verlag, 1985.

[Bee86] Michael J. Beeson. Proving programs and programming proofs. In *International Congress on Logic, Methodology and Philosophy of Science VII*, pages 51–82, Amsterdam, 1986. North-Holland. Proceedings of a meeting held at Salzburg, Austria, in July, 1983.

[Bee88] Michael J. Beeson. Towards a computation system based on set theory. *Theoretical Computer Science*, 60:297–340, 1988.

[Ben85] Ermanno Bencivenga. Free logics. In Gabbay and Guenthner [GG85], chapter III.6, pages 373–426.

[BJ89] George S. Boolos and Richard C. Jeffrey. *Computability and Logic*. Cambridge University Press, Cambridge, UK, third edition, 1989.

[BM79] R. S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, New York, NY, 1979.

[BM88] R. S. Boyer and J S. Moore. *A Computational Logic Handbook*. Academic Press, New York, NY, 1988.

[Bou68] N. Bourbaki. *Elements of Mathematics: Theory of Sets*. Addison-Wesley, Reading, MA, 1968.

[BP83] Paul Benacerraf and Hilary Putnam, editors. *Philosophy of Mathematics: Selected Readings*. Cambridge University Press, Cambridge, England, second edition, 1983.

[Bro67] Luitzen Egbertus Jan Brouwer. On the significance of the principle of excluded middle in mathematics, especially in function theory. In van Heijenoort [vH67], pages 334–345. First published 1923.

[Bry86] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[Bry92] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[BS89] G. Birtwistle and P. A. Subrahmanyam, editors. *Current Trends in Hardware Verification and Theorem Proving.* Springer-Verlag, New York, NY, 1989.

[CAB+86] R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Englewood Cliffs, NJ, 1986.

[Can55] Georg Cantor. *Contributions to the Founding of the Theory of Transfinite Numbers.* Dover Publications, Inc., 1955. Translated and with an 80 page introduction by Philip E. B. Jourdain. Originally published as *Beiträge zur Begründung der Transfiniten Mengenlehre*, 1895 and 1897.

[Can91] D. Cantone. Decision procedures for elementary sublanguages of set theory X: Multilevel syllogistic extended by the singleton and powerset operators. *Journal of Automated Reasoning*, 7(2):193–230, June 1991.

[Car58] Rudolf Carnap. *Introduction to Symbolic Logic and Its Applications.* Dover Publications, Inc., New York, NY, 1958. English translation of *Einführung in die symbolische Logik*, 1954.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[Che80] Brian F. Chellas. *Modal Logic: An Introduction.* Cambridge University Press, Cambridge, UK, 1980.

[Chu56] Alonzo Church. *Introduction to Mathematical Logic*, volume 1 of *Princeton Mathematical Series.* Princeton University Press, Princeton, NJ, 1956. Volume 2 never appeared.

[CJ90] J. H. Cheng and C. B. Jones. On the usability of logics which handle partial functions. In Carroll Morgan and J. C. P. Woodcock, editors, *Proceedings of the Third Refinement Workshop*, pages 51–69. Springer-Verlag Workshops in Computing, 1990.

[CKM+91] Dan Craigen, Sentot Kromodimoeljo, Irwin Meisels, Bill Pase, and Mark Saaltink. EVES: An overview. In Prehn and Toetenel [PT91], pages 389–405.

[CL73] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Computer Science and Applied mathematics. Academic Press, New York, NY, 1973.

[Cop67] Irving M. Copi. *Symbolic Logic.* The Macmillan Company, third edition, 1967.

[Cur77] Haskell B. Curry. *Foundations of Mathematical Logic.* Dover Publications, Inc., New York, NY, 1977.

[Dau88] Joseph W. Dauben. Abraham Robinson and nonstandard analysis: History, philosophy and foundations of mathematics. In Aspray and Kitcher [AK88], pages 177–200.

[Dav89] Ruth E. Davis. *Truth, Deduction, and Computation.* Principles of Computer Science Series. Computer Science Press, an imprint of W. H. Freeman and Company, New York, 1989.

[dB80] N. G. de Bruijn. A survey of the project Automath. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 589–606. Academic Press, 1980.

[Ded63] Richard Dedekind. *Essays on the Theory of Numbers.* Dover Publications, Inc., New York, NY, 1963. Reprint of the 1901 Translation by Wooster Woodruff Beman; the original essay was written 1887.

[DeL70] Howard DeLong. *A Profile of Mathematical Logic.* Addison-Wesley series in mathematics. Addison-Wesley, Reading, MA, 1970.

[DFH+91] Gilles Dowek, Amy Felty, Hugo Herbelin, Gérard Huet, Christine Paulin-Mohring, and Benjamin Werner. The COQ proof assistant user's guide: Version 5.6. Rapports Techniques 134, INRIA, Rocquencourt, France, December 1991.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 6, pages 243–320. Elsevier and MIT press, Amsterdam, The Netherlands, and Cambridge, MA, 1990.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[DS90] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and Program Semantics*. Texts and Monographs in Computer Science. Springer-Verlag, New York, NY, 1990.

[DST80] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpressions problem. *Journal of the ACM*, 27(4):758–771, October 1980.

[Dun85] J. Michael Dunn. Relevance logic and entailment. In Gabbay and Guenthner [GG85], chapter III.3, pages 117–224.

[Dun91] William Dunham. *Journey Through Genius: The Great Theorems of Mathematics*. Penguin Books, New York, NY, 1991.

[EFT84] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Undergraduate Texts in Mathematics. Springer-Verlag, New York, NY, 1984.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York, NY, 1972.

[Far90] William M. Farmer. A partial functions version of Church's simple theory of types. *Journal of Symbolic Logic*, 55(3):1269–1291, September 1990.

[FBHL84] A. A. Fraenkel, Y. Bar-Hillel, and A. Levy. *Foundations of Set Theory*, volume 67 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, The Netherlands, second printing, second edition, 1984.

[Fef89] Solomon Feferman. *The Number Systems: Foundations of Algebra and Analysis*. Chelsea Publishing Company, New York, NY, second edition, 1989. First edition published 1964.

[FGT93] William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. IMPS: An interactive mathematical proof system. *Journal of Automated Reasoning*, 11(2):213–248, October 1993.

[Fre67] G. Frege. Letter to Russell. In van Heijenoort [vH67], pages 126–128. Written 1902.

[Gal86] Jean H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper and Row, New York, 1986.

[Gar84] James W. Garson. Quantification in modal logic. In Dov M. Gabbay and Franz Guenthner, editors, *Handbook of Philosophical Logic–Volume II: Extensions of Classical Logic*, volume 165 of *Synthese Library*, chapter II.5, pages 249–307. D. Reidel Publishing Company, Dordrecht, Holland, 1984.

[GG83] Dov M. Gabbay and Franz Guenthner, editors. *Handbook of Philosophical Logic–Volume I: Elements of Classical Logic*, volume 164 of *Synthese Library*. D. Reidel Publishing Company, Dordrecht, Holland, 1983.

[GG85] Dov M. Gabbay and Franz Guenthner, editors. *Handbook of Philosophical Logic–Volume III: Alternatives to Classical Logic*, volume 166 of *Synthese Library*. D. Reidel Publishing Company, Dordrecht, Holland, 1985.

[GH93] John V. Guttag and James J. Horning with S. J. Garland, K. D. Jones, A. Modet, and J. M. Wing. *Larch: Languages and Tools for Formal Specification*. Texts and Monographs in Computer Science. Springer-Verlag, 1993.

[GM93] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic*. Cambridge University Press, Cambridge, UK, 1993.

[Gog89] J. A. Goguen. OBJ as a theorem prover with application to hardware verification. In Birtwistle and Subrahmanyam [BS89], pages 218–267.

[Gol88] Warren Goldfarb. Poincaré against the Logicists. In Aspray and Kitcher [AK88], pages 61–81.

[Gor88] Michael J. C. Gordon. *Programming Language Theory and its Implementation*. Prentice Hall International Series in Computer Science. Prentice Hall, Hemel Hempstead, UK, 1988.

[Gor89] Michael J. C. Gordon. Mechanizing programming logics in higher-order logic. In Birtwistle and Subrahmanyam [BS89], pages 387–439.

[GTWW77] Joseph Goguen, James Thatcher, Eric Wagner, and Jesse Wright. Initial algebra semantics and continuous algebras. *Journal of the ACM*, 24(1):68–95, January 1977.

[Gum89] Raymond D. Gumb. *Programming Logics: An Introduction to Verification and Semantics*. John Wiley and Sons, New York, NY, 1989.

[Hal60] Paul R. Halmos. *Naive Set Theory*. The University Series in Undergraduate Mathematics. Van Nostrand Reinhold Company, New York, NY, 1960.

[Hal84] Michael Hallett. *Cantorian Set Theory and Limitation of Size*. Number 10 in Oxford Logic Guides. Oxford University Press, Oxford, England, 1984.

[Hat82] William S. Hatcher. *The Logical Foundations of Mathematics*. Pergamon Press, Oxford, UK, 1982.

[Haz83] Allen Hazen. Predicative logics. In Gabbay and Guenthner [GG83], chapter I.5, pages 331–407.

[HC68] G. E. Hughes and M. J. Cresswell. *An Introduction to Modal Logic*. Methuen, London, UK, 1968.

[HC84] G. E. Hughes and M. J. Cresswell. *A Companion to Modal Logic*. Methuen, London, UK, 1984.

[HD92] F. K. Hanna and N. Daeche. Dependent types and formal synthesis. In C. A. R. Hoare and M. J. C. Gordon, editors, *Mechanized Reasoning and Hardware Design*, pages 121–135, Hemel Hempstead, UK, 1992. Prentice Hall International Series in Computer Science.

[HDL89] F. Keith Hanna, Neil Daeche, and Mark Longley. Specification and verification using dependent types. *IEEE Transactions on Software Engineering*, 16(9):949–964, September 1989.

[HF89] Joseph Y. Halpern and Ronald Fagin. Modeling knowledge and action in distributed systems. *Distributed Computing*, 3:159–177, 1989.

[Hil83] David Hilbert. On the infinite. In Benacerraf and Putnam [BP83], pages 183–201. First published 1926.

[HN88] Susumu Hayashi and Hiroshi Nakano. *PX: A Computational Logic*. Foundations of Computing. MIT Press, Cambridge, MA, 1988.

[Hoa69] C. A. R. Hoare. An axiomatic basis of computer programming. *Communications of the ACM*, 12(10):576–580, October 1969.

[Hod77] Wilfred Hodges. *Logic*. Penguin, London, UK, 1977.

[Hod83] Wilfred Hodges. Elementary predicate logic. In Gabbay and Guenthner [GG83], chapter I.1, pages 1–131.

[JM86] Farnam Jahanian and Aloysius Ka-Lau Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, SE-12(9):890–904, September 1986.

[Joh87] P. T. Johnstone. *Notes on Logic and Set Theory.* Cambridge Mathematical Textbooks. Cambridge University Press, Cambridge, UK, 1987.

[KK62] William Kneale and Martha Kneale. *The Development of Logic.* Clarendon Press, Oxford, England, 1962. Paperback edition, 1984.

[Kle52] Stephen C. Kleene. *Introduction to Metamathematics.* North-Holland, Amsterdam, The Netherlands, 1952.

[Kle67] Stephen C. Kleene. *Mathematical Logic.* John Wiley and Sons, New York, NY, 1967.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, November 1990.

[KZ88] D. Kapur and H. Zhang. RRL: A rewrite rule laboratory. In Lusk and Overbeek [LO88], pages 768–769.

[Lak78] Imre Lakatos. Cauchy and the continuum: the significance of non-standard analysis for the history of philosophy of mathematics. In John Worrall and Gregory Currie, editors, *Mathematics, Science and Epistemology: Philosophical Papers of Imre Lakatos, Volume 2*, chapter 3, pages 43–60. Cambridge University Press, Cambridge, UK, 1978. (Cambridge Paperback Library edition, 1990).

[Lam83] L. Lamport. Sometime is sometimes not never. In *10th ACM Symposium on Principles of Programming Languages*, pages 174–185, Austin, TX, January 1983.

[Lam91] Leslie Lamport. The temporal logic of actions. Technical Report 79, DEC Systems Research Center, Palo Alto, CA, December 1991. To Appear in ACM Transactions on Programming Languages and Systems.

[Lei69] A. C. Leisenring. *Mathematical Logic and Hilbert's ε-Symbol.* Gordon and Breach Science Publishers, New York, NY, 1969.

[Lem87] E. J. Lemmon. *Beginning Logic.* Chapman and Hall, London, UK, second edition, 1987.

[Lev79] Azriel Levy. *Basic Set Theory.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, Germany, 1979.

[LO88] E. Lusk and R. Overbeek, editors. *9th International Conference on Automated Deduction (CADE)*, volume 310 of *Lecture Notes in Computer Science*, Argonne, IL, May 1988. Springer-Verlag.

[McA89] David A. McAllester. *ONTIC: A Knowledge Representation System for Mathematics*. MIT Press, Cambridge, MA, 1989.

[McC90] W. W. McCune. OTTER 2.0 users guide. Technical Report ANL-90/9, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, March 1990.

[Men64] Elliott Mendelson. *Introduction to Mathematical Logic*. The University Series in Undergraduate Mathematics. D. Van Nostrand Company, New York, NY, 1964.

[Min92] Grigori Mints. *A Short Introduction to Modal Logic*, volume 30 of *CSLI Lecture Notes*. Center for the Study of Language and Information, Stanford, CA, 1992.

[Mon76] J. Donald Monk. *Mathematical Logic*. Graduate Texts in Mathematics. Springer-Verlag, New York, NY, 1976.

[Moo88] Gregory H. Moore. The emergence of first-order logic. In Aspray and Kitcher [AK88], pages 95–135.

[MR91] C. A. Middelburg and G. R. Renardel de Lavalette. LPF and MPL_ω—a logical comparison of VDM SL and COLD-K. In Prehn and Toetenel [PT91], pages 279–308.

[Mus80] D. R. Musser. Abstract data type specification in the Affirm system. *IEEE Transactions on Software Engineering*, 6(1):24–32, January 1980.

[NO79] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[NS93] Anil Nerode and Richard A. Shore. *Logic for Applications*. Texts and Monographs in Computer Science. Springer-Verlag, New York, NY, 1993.

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, pages 748–752, Saratoga, NY, June 1992. Volume 607 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag.

[Ost90] Jonathan S. Ostroff. A logic for real-time discrete event processes. *IEEE Control Systems Magazine*, 10(4):95–102, June 1990.

[Owe93] Olaf Owe. Partial logics reconsidered: A conservative approach. *Formal Aspects of Computing*, 5(3):208–223, 1993.

[Par93] David Lorge Parnas. Predicate logic for software engineering. *IEEE Transactions on Software Engineering*, 19(9):856–862, September 1993.

[Pau88] Lawrence C. Paulson. Isabelle: The next seven hundred theorem provers. In Lusk and Overbeek [LO88], pages 772–773.

[Pea67] Giuseppe Peano. The principles of arithmetic, presented by a new method. In van Heijenoort [vH67], pages 83–97. First published 1889.

[PH78] Jeff Paris and Leo Harrington. A mathematical incompleteness in Peano arithmetic. In Barwise [Bar78a], chapter D8, pages 1133–1142.

[Phi92] I. C. C. Phillips. Recursion theory. In S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science; Volume 1 Background: Mathematical Structures*, pages 79–187. Oxford Science Publications, Oxford, UK, 1992.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th Symposium on Foundations of Computer Science*, pages 46–57, Providence, RI, November 1977. ACM.

[Pra92] Sanjiva Prasad. Verification of numerical programs using Penelope/Ariel. In *COMPASS '92 (Proceedings of the Seventh Annual Conference on Computer Assurance)*, pages 11–24, Gaithersburg, MD, June 1992. IEEE Washington Section.

[PT91] S. Prehn and W. J. Toetenel, editors. *VDM '91: Formal Software Development Methods*, volume 551 of *Lecture Notes in Computer Science*, Noordwijkerhout, The Netherlands, October 1991. Springer-Verlag. Volume 1: Conference Contributions.

[Rab78] Michael O. Rabin. Decidable theories. In Barwise [Bar78a], chapter C8, pages 595–629.

[Ram90] F. P. Ramsey. The foundations of mathematics. In D. H. Mellor, editor, *Philosophical Papers of F. P. Ramsey*, chapter 8, pages 164–224. Cambridge University Press, Cambridge, UK, 1990. Originally published in *Proceedings of the London Mathematical Society*, 25, pp. 338–384, 1925.

[Ran88] P. Venkat Rangan. An axiomatic basis for trust in distributed systems. In *Proceedings of the Symposium on Security and Privacy*, pages 204–211, Oakland, CA, April 1988. IEEE Computer Society.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[Rob74] A. Robinson. *Nonstandard Analysis*. North-Holland, Amsterdam, The Netherlands, revised edition, 1974.

[RR63] H. Rubin and J. Rubin. *Equivalents of the Axiom of Choice*. North-Holland, 1963.

[Rud92] Piotr Rudnicki. *An Overview of the MIZAR Project*. Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, June 1992.

[Rus67] Bertrand Russell. Letter to Frege. In van Heijenoort [vH67], pages 124–125. Written 1902.

[Sch67] Moses Schönfinkel. On the building blocks of mathematical logic. In van Heijenoort [vH67], pages 355–366. First published 1924.

[Sco79] D. S. Scott. Identity and existence in intuitionistic logic. In M. P. Fourman, C. J. Mulvey, and D. S. Scott, editors, *Applications of Sheaves*, pages 660–696. Volume 753 of *Lecture Notes in Mathematics*, Springer-Verlag, 1979.

[Sha94] N. Shankar. *Metamathematics, Machines, and Gödel's Proof*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, UK, 1994.

[Sho67] Joseph R. Shoenfield. *Mathematical Logic*. Addison-Wesley, Reading, MA, 1967.

[Sho77] Robert E. Shostak. On the SUP-INF method for proving Presburger formulas. *Journal of the ACM*, 24(4):529–543, October 1977.

[Sho78a] J. R. Shoenfield. Axioms of set theory. In Barwise [Bar78a], chapter B1, pages 321–344.

[Sho78b] Robert E. Shostak. An algorithm for reasoning about equality. *Communications of the ACM*, 21(7):583–585, July 1978.

[Sho79] Robert E. Shostak. A practical decision procedure for arithmetic with function symbols. *Journal of the ACM*, 26(2):351–360, April 1979.

[Sho84] Robert E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, January 1984.

[Spi88] J. M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics.* Cambridge Tracts in Theoretical Computer Science 3. Cambridge University Press, Cambridge, UK, 1988.

[Sti81] Mark Stickel. A unification algorithm for associative-commutative functions. *Journal of the ACM*, 28(3):423–434, July 1981.

[Sti85] Mark E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1(4):333–355, December 1985.

[Sun83] Göran Sundholm. Systems of deduction. In Gabbay and Guenthner [GG83], chapter I.2, pages 133–188.

[Sup72] Patrick Suppes. *Axiomatic Set Theory.* Dover Publications, Inc., New York, NY, 1972.

[Tar76] Alfred Tarski. *Introduction to Logic and to the Methodology of Deductive Sciences.* Oxford University Press, New York, NY, third edition, 1976. First published 1941.

[Tho91] Simon Thompson. *Type Theory and Functional Programming.* International Computer Science Series. Addison-Wesley, Wokingham, England, 1991.

[TvD88] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics: An Introduction*, volume 121 and volume 123 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, Amsterdam, Holland, 1988. In two volumes.

[US93] Thomás E. Uribe and Mark E. Stickel. *Ordered Binary Decision Diagrams and the Davis-Putnam Procedure*, 1993. Submitted for publication.

[vBD83] Johan van Bentham and Kees Doets. Higher-order logic. In Gabbay and Guenthner [GG83], chapter I.4, pages 275–329.

[vH67] Jean van Heijenoort, editor. *From Frege to Gödel.* Harvard University Press, Cambridge, MA, 1967.

[WR27] A. N. Whitehead and B. Russell. *Principia Mathematica.* Cambridge University Press, Cambridge, revised edition, 1925–1927. Three volumes. The first edition was published 1910–1913.

[WR67] Alfred North Whitehead and Bertrand Russell. Incomplete symbols: Descriptions. In van Heijenoort [vH67], pages 216–223. First published 1910; Russell's original account "On Denoting" appeared in 1905.