P. Madhusudan

Logic in Computer Science

Rough course notes

November 5, 2020

Contents

1 Logic over Structures: A Single Known Structure, Classes of Structures, and All Structures 1
1.1 Logic on a Fixed Known Structure 1
1.2 Logic on a Fixed Class of Structures 3
1.3 Logic on All Structures 5
1.4 Logics over structures: Theories and Questions 7

2 Propositional Logic 11
2.1 Propositional Logic 11
2.1.1 Syntax 11
2.1.2 What does the above mean with ::=, etc? 11
2.2 Some definitions and theorems 14
2.3 Compactness Theorem 15
2.4 Resolution 18

3 Quantifier Elimination and Decidability 23
3.1 Quantifiier Elimination 23
3.2 Dense Linear Orders without Endpoints 24
3.3 Quantifier Elimination for rationals with addition: $(\mathbb{Q}, 0, 1, +, -, <, =)$ 27
3.4 The Theory of Reals with Addition 30
3.4.1 Aside: Axiomatizations 30
3.4.2 Other theories that admit quantifier elimination 31

4 Validity of FOL is undecidable and is r.e.-hard 33
4.1 Validity of FOL is r.e.-hard 34
4.2 Trakhtenbrot's theorem: Validity of FOL over finite models is undecidable, and co-r.e. hard 39

5 Quantifier-free theory of equality 43
5.1 Decidability using Bounded Models 43

5.2 An Algorithm for Conjunctive Formulas 44
5.2.1 Computing $CC(E)$ 47
5.3 Axioms for The Theory of Equality 49

6 Completeness Theorem: FO Validity is r.e. 53
6.1 Prenex Normal Form 55
6.2 Skolemization / Herbrandization 55
6.3 Herbrand's theorem 57
6.4 Some consequences of Herbrand's theorem 62
6.5 Gödel's completeness theorem: FO Validity is recursively enumerable 63
6.5.1 The case of finite sets of formulas 66
6.5.2 The case for infinite sets of formulas 67
6.5.3 Completeness Theorem 68
6.6 Observations and Consequences 70

7 Number Theory and Correctness of Programs: Incompleteness 73
7.1 Program Verification 75
7.2 Incompleteness of the theory of natural numbers with additional and multiplication 75
7.3 Further Remarks 79

Chapter 1
Logic over Structures: A Single Known Structure, Classes of Structures, and All Structures

1.1 Logic on a Fixed Known Structure

You are already familiar with logic on a fixed structure (or context or world). For example, you know what this statement means:

$$\forall x \in \mathbb{N} \ \exists y \in \mathbb{N} \ x < y$$

It says "for every natural number x, there exists a natural number y such that $x < y$. You have learned this in courses in discrete mathematics.

If you haven't or need a primer, I recommend reading it from the following sources:

- Madhu's primer for CS173: https://courses.grainger.illinois.edu/cs173/fa2017/B-lecture/NotesByMadhu/Notes-1.pdf

The main thing to note here is that you *know* the structure/universe/context you are talking about— in this case natural numbers.

You should make sure you know several things about such logical notation:

- You should know the Boolean connectives $\wedge$, $\vee$, and $\neg$.
- You should know the meaning of $\forall$ ("forall") and $\exists$ ("exists"), which are *quantifiers*.
- You should know that $\alpha \Rightarrow \beta$ (read α *implies* β) has a formal meaning that is precisely the same as $(\neg\alpha) \vee \beta$, and not some other uses in English. For example, "Goldilocks is the president of the United States implies all gorillas are green" is a statement that is true in the current world, since Goldilocks is not the president of the United States (I am assuming this is true when you read this as well).
- One really needs only the connectives $\vee$ and $\neg$; the rest can be derived or defined as shortcuts:
  - $\alpha \wedge \beta$ is logically equivalent to $\neg(\neg\alpha \vee \neg\beta)$
  - $\alpha \Rightarrow \beta$ is logically equivalent to $(\neg\alpha) \vee \beta$
  - $\alpha \Leftrightarrow \beta$ is logically equivalent to $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

- You should know the de Morgan laws and how negation goes into quantifiers:
  - $\neg(\alpha \vee \beta)$ is equivalent to $(\neg\alpha) \wedge (\neg\beta)$
  - $\neg(\alpha \wedge \beta)$ is equivalent to $(\neg\alpha) \vee (\neg\beta)$
  - $\neg(\exists x.\ \alpha)$ is equivalent to $\forall x.(\neg\alpha)$
  - $\neg(\forall x.\ \alpha)$ is equivalent to $\exists x.(\neg\alpha)$

  Using the above, you should know that you you can "push" the negations all the way in so that they are applied only to atomic symbols/formulae.

As a concrete example, let us discuss logics over a fixed structure— the set of natural numbers $\mathcal{N}$.

In propositional logic over a fixed structure (or universe or world), you would have a mapping between propositional symbols and *statements*. For example, p could be the statement "there are finitely many primes" and q could be the statement "all primes other than 2 are odd". Then p is false in this world, and q is true in this world, etc.

First order logic over natural numbers is more powerful and useful. Over natural numbers we have several functions and relations that we know. For example, $+$ is a (binary) function, $\times$ is a (binary) function, and *square* is a (unary) function. And $<, \leq$ are all (binary) relations. In fact, another relation that we overlook sometimes is the *equality* relation $=$, since it's so common.

Now, let us fix this signature of *symbols* for functions and relations. Let us also fix a set of symbols to denote *variables*, called *Var*. And define a first order logic formally:

$$\mathit{Terms:}\ \ t, t' ::= x \mid c \mid +(x, y) \mid \times(x, y) \mid \mathit{square}(x)$$

$$\mathit{Formulas:}\ \ \varphi, \varphi' ::= t = t' \mid t < t' \mid t \leq t' \mid \varphi \vee \varphi' \mid \varphi \wedge \varphi' \mid \neg\varphi \mid \forall x.\ \varphi \mid \exists x.\varphi$$

where $c \in \mathbb{N}, x \in \mathit{Var}$.

We have here two kinds of expressions— terms and formulas. Terms are obtained from constants and variables by applying functions, recursively, and intuitively "evaluate" to some number. For example, $+(5, 8)$ is a term. Note that we write functions like $+$ with the symbol in front, rather than in infix notation; i.e., we write $+(5, 8)$ instead of $5+8$. But as you are familiar with computer science, you can think of $+$ as a function that you "call", and so $+(5, 8)$ should make sense. $+(9, x)$ is a term as well. And $\mathit{square}(*(13384398, \mathit{square}(y)))$ is a term too. Clearly there are infinitely many terms.

We don't really need all constants in the grammar. 0 and 1 are sufficient, as any other constant can be expressed as an appropriate sum of 1s.

Formulas, on the other hand, evaluate to a Boolean, i.e., *true* or *false*. The *atomic* formulas are those that are formed from terms— those of the form $t = t'$, $t < t'$ and $t \leq t'$. Formulas are basically atomic formulas closed under Boolean operations and quantification.

Note that we write $\forall x.\ \exists y.\ \ x < y$, rather than $\forall x \in \mathbb{N}.\ \exists y \in \mathbb{N}.\ \ x < y$, since the universe over which variables are quantified are assumed to be natural numbers anyway. If there are multiple sorts of objects that we want to quantify over, we would introduce the more general syntax where every time we quantify, we will say which sort it is over (or designate different sets of variables for each sort). This is similar to programming, where we declare variables to be of different sorts— like integers or strings.

You should be able to read and write FO formulas over natural numbers. For instance, the formula $\forall x.\ (x > 2) \Rightarrow \mathit{square}(x) > x$ says “the square of any number greater than two is larger than it”, which is happens to be true over natural numbers. Similarly, if we wanted to say “every number other than 0 has a smaller number”, we would write this as $\forall x. \neg(x = 0) \Rightarrow (\exists y.\ (y < x))$, which also happens to be true over natural numbers.

Now consider the formula $\forall x.(x > y)$. Is it true over natural numbers? It’s hard to imagine how to interpret the statement as its unclear what y is. This leads us to define something called sentences.

A *sentence* is a formula where every variable is quantified, We will define this formally later, but it should be clear to you what this means.... any variable that occurs in the formula must have a quantification “outside” it such that the variable is in scope of that quantifier. This is similar to programming— we may require all variables used in a program to be *declared*, i.e., every use of a variable should be in the scope of a declaration of it. Sentences evaluate to true or false on a structure.

The first-order *theory* of the structure $\mathbb{N}$ is the set of FO sentences that hold in that model, denoted $Th(\mathbb{N})$. Note that since any sentence α must either be true or false in the structure, and hence either α or $\neg\alpha$ must be in the theory. More generally, a theory is just a set of sentences. And a theory is said to be *complete* if for every sentence α, either α or $\neg\alpha$ belongs to it. So $Th(\mathbb{N})$ is complete (indeed, the theory of any fixed structure is complete).

1.2 Logic on a Fixed Class of Structures

In mathematics and computer science, we often want to express properties that hold on a *class* of structures, not just a single structure. For example, we may ask what formulas/sentences are true over *groups*, or over *finite graphs*, or over *trees*, or over *linked lists*, or over *recursively defined datatypes*, or over *SQL (relational) databases*, or about *objects* in a class.

Unlike a single structure, like $\mathbb{N}$ or $\mathbb{Z}$ or $\mathbb{R}$, we are interested in a *class* of structures.

Groups

Let us take groups. A group is defined as a set S endowed with a binary relation $\circ : S \times S \to S$ that satisfies the following properties:

Associativity: for every $a, b, c \in S$, $(a \circ b) \circ c = a \circ (b \circ c)$
Identity: There is an element $e \in S$ such that for every $a \in S$, $a \circ e = e \circ a = a$.
Inverse: For every $a \in S$, there exists an $a' \in S$ such that $a \circ a' = a' \circ a = e$.

For example, the set of integers $\mathbb{Z}$ with the operator $+$ forms a group (0 is the identity, and for every $i \in \mathbb{Z}$, $-i$ is its inverse. Another class of examples of groups is obtained by taking a finite set E and considering the set of elements consisting of permutations of E, with the binary operation being *composition* of permutations (a composition of permutations is a permutation as well). The identity permutation is the identity element, and every permutation has an inverse, of course, which "reverts" the permutation.

Now, there are of course properties that hold on *all* groups. For instance, the identity element must be unique. Here's a proof: Assume $e, f \in S$ are both identity elements. Then $e.f = e$ (since f is an identity) and $e.f = f$ (since e is an identity). Hence $e = f$.

Similarly, one can show that every element's inverse is unique.

The above proof shows that the fact that 0 is the unique identity element for $+$ over $\mathbb{Z}$ is not a particular property satisfied only on integers, but rather is a property shared among all groups. The field of group theory studies groups in their own right, since they occur commonly in many areas and applications.

We can now define the *first order theory* of groups as the set of all FO sentences that hold over groups. The signature could include a special constant e to denote the identity element. The sentence $\forall a, b, c.(a \circ b = e \wedge b \circ a = e \wedge a \circ c = e \wedge c \circ a = e) \Rightarrow b = c$, which says that every element has a unique inverse, is hence a theorem in this theory.

Note however that the theory of groups is not a complete theory. For example, the sentence $\forall x, y.x \circ y = y \circ x$ is not a theorem nor is its negation a theorem. There are some groups where this property is true (like $+$ over integers) and some where it's not true (like permutations of a fixed finite set of elements).

Graphs

Graphs are ubiquitous in computer science. Each graph can be seen as a model where there is a set/universe which is *finite* and that has a *binary relation* E over it, which models the set of edges. We would expect E to be *symmetric* (if $E(u, v)$ holds, then $E(v, u)$ also holds). And we don't want "self-edges" (for any u, $E(u, u)$ does not hold).

One can then defined the *theory of graphs*— which consists of all FO sentences true over graphs. This FO theory is not terribly interesting, as there are very few properties about graphs you can express using just FO theory on graphs. Note that this theory is not complete, again, of course— the sentence $\forall u, v.E(u, v)$ is neither true in all graphs nor false in all graphs.

One can of course define any subclass of graphs— such as planar graphs or bipartite graphs— and talk about the theory of such subclasses as well. How does the theory of graphs, TG, and the theory of planar graphs, TPG, compare? Is one a subset of the other? It is easy to see that any sentence that holds for all graphs is certainly true for all planar graphs as well. Hence $TG \subseteq TPG$.

In general, if C and $\mathcal{D}$ are two classes of structures with C a subclass of $\mathcal{D}$, then the theory of $\mathcal{D}$ would be a subset of the theory of C. The smaller the class, the larger its theory! In the limit, when there is only one structure, the theory becomes complete (and of course cannot get any larger without containing contradictions).

One can now ask— do we really need to have a single structure in order to obtain a complete theory? In other words, is there a class of structures/models C that has at least two structures such that its first order theory is complete? Strangely, the answer is *yes*! We will see examples of this in the course. For example:

- The theory of rationals with only the relations = and ≤ is the same as the theory of reals with the relations = and ≤! In other words, there is no *first-order sentence* that can distinguish between these structures.
  Note that the above is very specific to the fact that we have only FOL and only the fixed signature involving ≤. For example, if we had the function symbol $square$ that returns the square of a rational/real, then we *can* distinguish the two structures. (Can you come up with one? If you have in addition constants such as $0, 1, 2, \ldots$, it would be simpler.)
- More generally, the theory of dense linear orders without endpoints (no least or largest element) is complete. No matter which dense linear order you pick, you will find that the theory is the same! So here is an example of an infinite class of structures which no first order formula (with only ≤ in the signature) can distinguish.
- One extremely surprising result is that there are structures that are *non-isomorphic* to natural numbers and yet satisfy the same first-order properties of natural numbers! You can imagine an alien species having such a *non-standard* model of arithmetic in their head (though I wonder what kind of evolutionary circumstance would give rise to such models in their psyche), and yet we would agree with them about all theorems in FOL over arithmetic!

1.3 Logic on All Structures

Finally, we can consider logics on *all structures*. This may sound a bit unnatural and not very useful. But as we shall see, it is quite useful, as it gives a way to study general metatheorems in logic that are independent of a particular structure or class of structures.

One reason why logics over all possible structures is that one can *carve* out useful and natural fragments using *axiomatizations*. Axiomatizations are *logical* mechanisms of specifying a class of structures that you want to study.

Axiomatizations are, in a certain sense, the purest form of reasoning about a class of structures. If we want to reason about a class of structures $\{C\}$, then it remains how to *define* them formally so that we can reason with them. For instance, you and I may think we know what natural numbers and arithmetic are, but to formally reason with arithmetic, we must be able to state our assumptions clearly. If you start an argument with "There are infinitely many even numbers, and ..." and I interrupt you and ask you why that is so and I don't believe it, then you may provide a proof of it. But every proof you give will make assumptions (hopefully simpler assumptions), till at some point you give up and say that those are self obvious. For example, you may refuse to give a proof of why "$x + 0 = x$". So it is natural to ask whether there are some fundamental and simple assumptions about natural numbers that we all can agree upon (and people who don't believe them can go climb a gum tree) such that all arguments can be made only using such assumptions.

Presburger arithmetic is a particular set of axioms that characterize natural numbers with addition only. The axioms are:

(A1) $\forall x.\neg(x + 1 = 0)$
(A2) $\forall x, y.(x + 1 = y + 1) \Rightarrow x = y$
(A3) $\forall x.x + 0 = x$
(A4) $\forall x, y.x + (y + 1) = (x + y) + 1$
(A5) For any first order formula P with a free variable x, the following holds:

$$(P(0) \wedge \forall x.(P(x) \Rightarrow P(x + 1))) \Rightarrow \forall y.P(y)$$

The meaning and soundness of axioms (A1)–(A4) should be obvious. (A5) is actually a *set* of axioms, called an axiom schema. It says that for any property P of natural numbers expressible in FOL, induction is a sound way of proving that P holds on all natural numbers. More precisely, if it was true that P holds for 0 and for every x, if P holds for x then P holds for $x + 1$, then P must hold for all natural numbers.

It is an amazing fact that all first-order properties of natural numbers with addition can be proved just using the axioms above. We will not show this in this course, however, but argue a related theorem to show that the theory of natural numbers with addition is decidable. (It will become clear later why these are related.)

Perhaps a more natural example of axiomatizations is the characterization of groups. Recall that groups satisfy a specific set of properties (associativity, existence of identity, and existence of inverses). This is *the* definition of groups— we do not have some other "natural" model of groups in our minds. And furthermore it turns out that we can characterize the properties of groups in FOL, with the signature containing = and the sole binary function $\circ$, and an identity element (constant symbol) e:

(A1) Associative law: $\forall x, y, z.\ x \circ (y \circ z) = (x \circ y) \circ z$
(A2) Identity: $\forall x.\ (x \circ e = x \wedge e \circ x = x)$
(A3) Existence of inverses: $\forall x.\ \exists y.\ (x \circ y = e \wedge y \circ x = e)$

Note that in the above, the set of axioms is *finite* as opposed to Presburger arithmetic. Similar to the above, we can characterize many classes of algebraic structures— abelian groups, rings, fields, Boolean algebras, etc.

Given a set of axioms $\mathcal{A}$, we can think of it as culling out a class of structures from the class of all structures, namely those that satisfy the axioms. We can then talk about the *theory* of the axioms— the set of sentences that are satisfied in every structure in this culled out class of structures. Let $Th(\mathcal{A})$ denote the theory defined by the structures that satisfy the axioms $\mathcal{A}$.

For example, for the set of axioms defining groups above, its theory contains the statement: $\forall x, y, z.\ (x \circ y = e \wedge y \circ x = e \wedge x \circ z = e \wedge z \circ x = e) \Rightarrow (y = z)$

1.4 Logics over structures: Theories and Questions

Given the above discussion, we have four kinds of theories:

- The theory of a single structure M, denoted $Th(M)$.
- The theory of a class of structures C, denoted $Th(C)$.
- The theory of all structures, which we will call tautologies or valid sentences, denoted $Th(FOL)$.
- The theory of a set of axioms $Th(\mathcal{A})$, which the the theory of the class of structures that satisfy $\mathcal{A}$.

Given the above, one can ask many interesting questions and make some simple observations:

- Let us call a theory complete if for every sentence α, either α is in the theory or $\neg\alpha$ is in the theory.
- Note that the theory of a single structure M is always complete. However, the theory of a class of structures need not be complete. (For example, over groups, the sentence $\forall x, y. x = y$ is neither in the theory nor is its negation in the theory). Similarly, the theory of a set of axioms need not be complete.
- The theory of a structure or a nonempty class of structures can never have a contradiction— i.e., it cannot have both α and $\neg\alpha$. You can have one only or the other or neither, but not both. The theory of the empty class of structures contains all sentences, and hence contradictions.
- The theory of a set of axioms can have a contradiction. But this happens only if they define an *empty* class of structures. Otherwise, the theory will have no contradiction. (For example, if axioms include $\forall x.P(x)$ and $\exists x.\neg P(x)$, then the set of structures defined by these axioms is empty, and its theory contains all sentences.)
- If a (nonempty) class of structures C is a subclass of a class of structures $\mathcal{D}$, then $Th(\mathcal{D}) \subseteq Th(C)$. So as the class of structures get smaller, its theory gets larger. (In the limit, the class of structures is a single structure and the theory becomes largest and is complete.
- What is the complexity of deciding whether a sentence is valid (on all structures)? Is it decidable? If not, is it recursively enumerable? What is its precise complexity?
- Is there a set of axioms (some regular simple set of axioms) that characterize natural numbers with addition? Integers with addition? Natural numbers or integers with addition and multiplication? What about rationals with addition and/or multiplication? Reals?

- Is it possible to decide if a set of axioms has a contradiction? I.e., whether there is at least one structure satisfying it?
- For each of the above theories, independent of whether they can be axiomatized or not, are the theories decidable? (Can we build programs that check whether a theorem is true or not, i.e., belongs to the theory or not, completely automatically?)
- How do we know that proofs even exist? Can it be the case that there are (natural) classes of structures for which some theorems do not have proofs?
- If we fix a set of axioms, is it true that every theorem (statement in its theory) has a proof, always, that follow from the axioms? Maybe first we need to define what a proof is? What's a proof?

I encourage you to think of all such combinations of questions. Some will be trivial and you will be able to answer them. Most others you will be able to answer at the end of the course. These questions that will occupy us for roughly half of this course.

Here is a sample of the remarkable theorems in logic you will learn in this course:

- The set of all valid sentences (the set of sentences that are true in all structures) is not decidable. However, it is recursively enumerable!
- One can in fact set up a formal proof mechanism for proving valid sentences. Proofs are syntactic objects that (a) are finite and (b) can be verified easily using syntactic rules. And then we can show that any valid sentence has a proof! (This is Gödel's completeness theorem.) It then follows that a Turing machine/program can simply look for such proofs, and hence the set of valid formulas is recursively enumerable.
- More remarkably, if I have a set of axioms $\mathcal{A}$ where $\mathcal{A}$ is a finite set of axioms, then the theory of $\mathcal{A}$ is also recursively enumerable. This even holds if $\mathcal{A}$ is infinite and is a computable (or even recursively enumerable set). In fact, one can set up generic proof systems that work by assuming any set of axioms that can prove any theorem in the theory of the set of axioms. This is remarkable as it shows that any axiomatizable theory has proofs and a computer can just look for such proofs!
- If a set of axioms define a *complete* theory, then the theory is even *decidable*! There is a program that can take a statement and decide whether it is a theorem or not. (Don't ask me how long this will take, though!)
- There are several specific structures and signatures where you can ask whether its theory is decidable (naturals, integers, rationals, reals with addition and/or multiplication, etc.). Most of these have been settled. Note that a complete axiomatization of them also implies decidability, and hence undecidability of the theory means there is no complete axiomatization.
  Four remarkable results are:
  - The theory of natural numbers with addition is decidable (Presburger arithmetic above is a complete axiomatization).
  - The theory of natural numbers with addition and multiplication is *not* decidable. It follows that this theory hence does not have a complete axiomatization.

  This is essentially one of Gödel’s incompleteness theorems. It turns out that even validity of purely universally quantified formulas or purely existentially quantified formulas is undecidable.
  - The theory of reals with addition and multiplication is decidable!
  - The theory of rationals with addiion and multiplication is undecidable.

- Checking whether a set of axioms has no contradiction is undecidable.

Chapter 2
Propositional Logic

Abstract Propositional logic; satisfiability; validity; satisfiability in decidable and NP-complete; compactness

2.1 Propositional Logic

Use the template *chapter.tex* together with the document class SVMono (monograph-type books) or SVMult (edited books) to style the various elements of your chapter content conformable to the Springer Nature layout.

2.1.1 Syntax

Fix a *countable* set of proposition symbols $\mathcal{P}$.

The set of propositional formulas over $\mathcal{P}$ is defined as:

$$\varphi, \varphi' ::= p \mid (\varphi \vee \varphi') \mid (\neg\varphi)$$

where $p \in \mathcal{P}$

2.1.2 What does the above mean with ::=, etc?

Define the following sets, parameterized by $i \in \mathbb{N}$:

$$S_0 = \mathcal{P}$$

$$S_{i+1} = S_i \cup \{(\phi \vee phi') \mid \phi, \phi' \in S_i\} \cup \{(\neg\phi) \mid \phi \in S_i\}$$

Now set S to be:

$$S = \bigcup_{i \in \mathbb{N}} S_i$$

Note that the above is an infinite union. You can think of S as the set $S_0 \cup S_1 \cup S_2 \cup \ldots$. Or think of S as the set of elements e such that there is some $i \in \mathbb{N}$ such that $e \in S_i$:

$$S = \{e \mid \exists i \in \mathbb{N}.\ e \in S_i\}$$

In any case, S is well-defined as a set. This set S of expressions is what the grammar defines. Each S_i denotes the expressions that can be derived by the grammar in i steps. And the set of expressions are those that can be derived in some finite number of steps.

Given the above meaning, it's natural to prove properties about the set of expressions S using induction on i. More precisely, if we want to show a property P is true about S, then we:

- Establish P to be true for every expression in S_0.
- For every $i > 0$, we assume that P holds for every expression in S_{i-1} and prove it holds for all expressions in S_i.

The above is clearly a valid proof that P holds on S_i, for every $i \in \mathbb{N}$, and hence P holds on every expression in S.

2.1.2.1 Aside: Other ways to define the set

Another way to define the set defined by the grammar is that it is the *smallest* set T that satisfies the following properties:

(a) T contains p, for every $p \in \mathcal{P}$,
(b) If φ and φ' belong to T (they need not be different, of course), then so does $(\varphi \vee \varphi')$, and
(c) If φ belongs to T, then so does $(\neg\varphi)$.

Why is the above well-defined? In other words, why is there a unique single minimal set that satisfies the above conditions? Remember, sets can be incomparable (with respect to the $\subseteq$ relation).

First, is there even one set that satisfies it? Sure, take all possible "strings" involving propositions, and the symbols "(", ")", $\vee$ and $\neg$... that surely satisfies all the conditions, trivially! Second, why should there be a smallest? Well, first convince yourself that if a *set* of sets satisfy the above conditions, then the *intersection* will satisfy the conditions as well. (Why?) Clearly, the intersection of *all* sets that satisfy the conditions is a set that satisfies the conditions and is also the smallest set (with respect to inclusion). So the "smallest set satisfying the conditions" exists and is unique. That is why it is well-defined. There is a minimal set satisfying the above conditions and there is only one minimal set satisfying the above conditions.

Let us give a formal proof that the smallest set T as defined above is in fact the set of expressions S defined using the sets S_i. In other words, let us prove that S satisfies the required properties and is also the smallest set amongst all sets that satisfy the required properties.

First, let us show that S satisfies the required properties. Observe that for any i, $S_i \subset S_{i+1}$, since the definition of S_{i+1} explicitly includes S_i. Hence the sets increase with index, and hence $S_i \subseteq S_j$ for any $i \leq j$ (we won't prove this formally here; but you should! Use induction.). Now,

- Note that for any $p \in \mathcal{P}$, $p \in S_0$, and hence $p \in S$.
- Next, assume $\alpha, \beta \in S$. Then it must be the case that $\alpha \in S_i$ and $\beta \in S_j$ for some $i, j \in \mathbb{N}$. Let $k = max(i, j)$. Then S_k includes both α and β. Hence S_{k+1} will contain $(\alpha \vee \beta)$. Hence $(\alpha \vee \beta) \in S$.
- Now, assume $\alpha \in S$. Then $\alpha \in S_i$, for some $i \in \mathbb{N}$. Hence $(\neg\alpha) \in S_{i+1}$. Hence $(\neg\alpha) \in S$.

QED.

Now let us prove that S is the *minimal* set that satisfies the required properties. Let U be any set that satisfies the required properties. We want to prove that $S \subseteq U$. We will prove by induction on i that $S_i \subseteq T$ for every $i \in \mathbb{N}$. This clearly establishes that $S \subseteq U$.

The base case is when $i = 0$. Clearly $S_0 = \mathcal{P} \subseteq U$ since U is required to contain $\mathcal{P}$. Now, for the induction step. Let $i \in \mathbb{N}$ and $i > 0$. The induction hypothesis is that $S_j \subseteq U$ for any $j < t$. Now, let $\varphi \in S_i$ be an arbitrary element int S_i. Then there are three cases, by the definition of S_i. The first is that $\varphi \in S_{i-1}$: then clearly by the induction hypothesis, $\varphi \in U$. The second case is that $\varphi = (\alpha \vee \beta)$, where $\alpha, \beta \in S_i$. Then, again by induction hypothesis, $\alpha, \beta \in U$. But since U satisfies the required properties, $(\alpha \vee \beta) \in U$, i.e., $\varphi \in U$. The third case is when $\varphi = (\neg\alpha)$, where $\alpha \in S_i$. By the induction hypothesis, $\alpha \in U$. Since U satisfies the required closure property with respec to $\neg$, $\varphi = (\neg\alpha) \in U$. QED.

The above show that S, as defined above, exists, satisfies the required properties, and is the least set satisfying the required properties. Hence the two definitions, S and T, coincide. You can easily generalize this argument to any formal grammar (not just context-free grammars).

In general, if one defines a set as the smallest set S that satisfies conditions of the form "if these elements belong to S than these other elements must belong to S", then the smallest set is well-defined. But if you have conditions also saying "if these elements belong to S, these elements should *not* belong to S," then that may not be a well-defined set.

Further notes: (okay to ignore for now

The above way of defining sets or other thingamies *recursively* is very useful in computer science, in general. Reasoning about these sets/thingamies is done naturally using some form of *induction*. In fact, the reason why induction is useful in computer science is perhaps because many interesting classes can be defined using recursion.

Here are some other recursive definitions:

- Consider the operational semantics of a program— which states/configurations can a program reach? This set is best defined recursively. It is the smallest set S of states such that (a) S contains the initial states of the program, and (b) if a state s is in S and the program can transition in one step from s to s', then $s' \in S$.
- Consider *lists*. Lists support a *cons* operator that constructs an element onto a list (adds the element as the first element of the list). Lists over the elements E can then be defined as the smallest set S such that (a) S contains Nil, the empty list, and (b) if $l \in S$ and $e \in E$, then $cons(e, l)$ is in S as well.
- The set of *natural numbers* can itself be defined recursively. Let us denote by $succ$, the successor function (intuitively, given a number n, its successor is the next number, i.e., $n + 1$; however, this interpretation is only in our minds; $succ$ is just some function). Then the set of natural numbers is the smallest set S such that (a) S contains 0, and (b) for every $n \in S$, *succ(n)* belongs to S as well.

Indeed, the induction proof technique is heavily used to prove properties of all the above objects. For example, when showing a program does not throw an exception, one needs to prove that all reachable states of the program are states where exceptions don't happen, and we prove these typically in program verification using induction.

There are more general ways of defining sets or other thingamies such that there is a notion of "least" set/thingamie always exists, and hence gives a well-defined definition. One way we will see later is the notion of *monotonic* functions on a lattice, which always have *least fixpoints*, due to a general theorem called the Tarski-Knaster theorem.

The reason the former always gives a well-defined set is because the conditions can be seen as "monotonic functions" and a general theorem, called the Tarski-Knaster theorem, which says that least fixpoints always exist. We will return to such a concept later in the course.

The above is the reason why we define *grammars* in computer science using the above notation. The Backus-Naur Form (BNF) is often used in computer science to give well-formed expressions (like all well-formed programs in a programming language). These ways of defining grammars go back to Panini, who defined it for Sanskrit, in 5th century BCE!

Note that the grammar could have said $(\varphi \vee \varphi)$, instead of $(\varphi \vee \varphi')$; that will still define the same set. Just because we repeat φ doesn't mean that it must be the same. In formal context free grammars, φ is a *nonterminal* generating a language.

2.2 Some definitions and theorems

Fix a set of propositions $\mathcal{P}$ (finite or countably infinite).

A *model* or *valuation* is a function $v : \mathcal{P} \to \{T, F\}$.

The notion of whether a formular φ holds in a model/valuation v is denoted $v \models \varphi$. (And $v \not\models \varphi$ denotes that the formula φ does not hold under the valuation v). This notion is the natural one you know, and is defined formally as follows, recursively:

- $v \models p iff v(p) = T$, for any $p \in \mathcal{P}$
- $v \models (\alpha \vee \beta)$ iff $v \models \alpha$ or $v \models \beta$
- $v \models \neg\alpha$ iff $v \not\models \alpha$

Satisfiability and Validity:

A formula α is said to be *satisfiable* if there is some valuation v such that $v \models \alpha$.

A formula α is said to be *valid* if for every valuation v, $v \models \alpha$ holds. We write $\models \alpha$ to denote α is valid.

The following is easy to see:

Lemma 2.1 *A formula α is valid iff $\neg\alpha$ is not satisfiable.*

Also, of course, a formula α is satisfiable iff $\neg\alpha$ is not valid.

Two formulas α and β are said to be *equivalent*, denoted $\alpha \equiv \beta$, if for every valuation v, $v \models \alpha$ iff $v \models \beta$.

Relevance Lemma:

The relevance lemma says that whether φ holds under a valuation depends only on how the valuation maps the propositions that *occur* in the formula. This is intuitively obvious; surely, whether $(p \wedge q) \vee r$ holds is independent of whether the proposition s is mapped to true/false.

Let us define this formally. Let's first define which propositional occur in a formula, recursively, as $occ : F \rightarrow 2^{\mathcal{P}}$, where F denotes the set of all well-formed formulae:

- $occ(p) = \{p\}$
- $occ((\alpha \vee \beta)) = occ(\alpha) \cup occ(\beta)$
- $occ((\neg\alpha)) = occ(\alpha)$

We can now state the relevance lemma:

Lemma 2.2 (Relevance Lemma) *Let α be a formula and let $O = occ(\alpha)$ be the set of propositions that occur in it. Let v, v' be two valuations such that for every $p \in O$, $v(p) = v'(p)$. Then*

$$v \models \alpha \quad \textit{iff} \quad v' \models \alpha$$

Proof Proof is fairly easy and is left as an exercise. Prove by induction on structure of α. □

2.3 Compactness Theorem

Let S be a (finite or infinite) set of propositional formulas.

S is said to be *satisfiable* if there is a valuation v under which every formula in S holds.

S is said to be *finitely satisfiable* if every finite subset T of S is satisfiable.

Note that if S is satisfiable, then it is, of course, finitely satisfiable. Is the converse true? The compactness theorem says that it is!

Let us first show a lemma, which says that a finitely satisfiable set can always be extended by a formula or its negation, and at least one of these will result in a finitely satisfiable set.

Lemma 2.3 *Let S be finitely satisfiable. Let α be any formula. Then either $S \cup \{\alpha\}$ is finitely satisfiable or $S \cup \{\neg\alpha\}$ is finitely satisfiable.*

Proof Assume the contrary. Then $S \cup \{alpha\}$ and $S \cup \{\neg\alpha\}$ are both not finitely satisfiable. Since $S \cup \{\alpha\}$ is not finitely satisfiable, there is a finite subset U of S that is unsatisfiable. Similarly, since $S \cup \{/\!\alpha\}$ is not finitely satisfiable, there is a finite subset V of S that is unsatisfiable. Now consider $(U \cup V) \setminus \{\alpha, \neg\alpha\}$. Since $U \cup V \subseteq S$ and is finite and S is finitely satisfiable, $U \cup V$ is satisfiable. Let v be a model that satisfies $U \cup V$. Then either $v \models \alpha$ or $v \models \neg\alpha$ must hold. But this means that U is satisfiable or V is satisfiable. Which contradicts our assumption for U and V. □

Note that the above does not say how we can determine whether $S \cup \{\alpha\}$ is satisfiable or $S \cup \{\neg\alpha\}$ is finitely satisfiable. S can be infinite and deciding anyway may not make sense. But we know one of these extensions is finitely satisfiable.

We now turn to proving the compactness theorem. Let S be a finitely satisfiable set. We are going to grow S ever so slowly using propositions or negations of propositions (which are both formulae), keeping it finitely satisfiable. Once we are done handling every proposition, we would get an entire valuation for all propositions. We can then argue that this valuation will satisfy every formula in S, and hence S is satisfiable.

Theorem 2.1 (Compactness of propositional logic) *Let S be a set of formulas. Then S is satisfiable iff S is finitely satisfiable.*

Proof The direction showing that if S is satisfiable, then it is finitely satisfiable is trivial: if S is satisfiable, then there is a valuation v such that all formulas in S are true under v. Then clearly all finite subsets of S are also true under v, and hence each such finite subset is satisfiable.

For the other direction, assume S is finitely satisfiable. We are going to build a valuation v such that all formulas in S are satisfied under v.

Let us fix an enumeration of all propositions: $p_0, p_1, p_2, \ldots$; this is possible since it is a countable set.

Now, let us define the sets X_i, where $i \in \mathbb{N}$ as follows.

$$X_0 = S$$

$$\begin{aligned} X_i &= X_{i-1} \cup \{p_i\},\ \mathit{if} X_i \cup \{p_i\}\ \textit{is finitely satisfable} \\ &= X_{i-1} \cup \{\neg p_i\}, \mathit{otherwise} \end{aligned}$$

Now let $X = \cup_{i \in \mathbb{N}} X_i$.

First, we claim that each X_i is finitely satisfiable, by induction. $X_0 = S$ is finitely satisfiable. Let $i > 0$. Then if $X_{i=1} \cup \{p_i\}$ is finitely satisfiable, then X_i is equal to this set, and is hence finitely satisfiable. If not, X_i is set to $X_{i-1} \cup \{\neg p_i\}$, which is finitely satisfiable by the lemma we proved above (since X_{i-1} is finitely satisfiable and $X_{i-1} \cup \{p_i\}$ is not finitely satisfiable.

Second, we claim that X is also finitely satisfiable. Here's a proof. Let U be a finite subset of X. Then since each X_i is a subset of X_{i+1}, and since every element of U occurs first in some X_j, we can take the maximum of such indices to argue that there is some k such that $U \subseteq X_k$. Since X_k is finitely satisfiable (by claim we proved above), U is finitely satisfiable as well.

We are now ready to pull out the valuation v using X. First, notice that for any p, both p and $\neg p$ cannot belong to X (otherwise X won't be finitely satisfiable). Also, for any p, either p or $\neg p$ must belong to X (since at step i, we decided to throw either p or $\neg p$ into X_i). So we simply have v assign p to true or false depending on whether p or $\neg p$ is in X, respectively:

$$v(p) = T \;\; \mathit{iff} \;\; p \in X$$

We are now going to show that v satisfies every formula in S. Let $\alpha \in S$. Let Q be the finite set of propositions that occur in α. Let $j \in \mathbb{N}$ be a large enough index such that all propositions in Q have been considered when building the X_m sets. More precisely, let $Q = \{p_{r_1}, \ldots p_{r_m}\}$. Then choose $j = max\{r_1, \ldots, r_m\}$.

Let $R = (X \cap Q) \cup (X \cap \{\neg q \mid q \in Q\}$. In other words, R gathers, for each propositions in Q, either q or $\neg q$, depending on which occurs in X.

Now consider $R \cup \{\alpha\}$. Then $R \cup \{\alpha\} \subseteq X$ and is finite. Hence R is satisfiable. Let v' be some valuation that satisfies $R \cup \{\alpha\}$. But then v' must map a proposition $qinQ$ to true iff $q \in R$ iff $q \in X$. Hence v' agrees with v on all propositions in Q. Hence, by the Relavance Lemma, since α holds under v', α holds under v as well. So we have shown that every formula in S is satisfied under v. Hence S is satisfiable. □

One of the powers of the compactness theorem is that we can apply it when interpreting every proposition in some way *in our head*. It roughly says that if we can formulate a set of propositions (with their true meaning only in our minds) but capture the relationships between these propositions as an infinite set of propositional formulae, and if every finite subset is satisfiable then the whole set is satisfiable.

We will apply the compactness theorem in various ways. In one exercise, we will show that all infinite planar graphs are 4-colorable because every finite planar graph is 4-colorable. And when dealing with FOL, we will prove crucial theorems there using the compactness theorem for propositional logic, interpreting propositions as more complex things in the FO world.

I encourage you to also see the Madhavan-Suresh notes for a proof of compactness theorem using König's Lemma. It is essentially the same proof, but presented using a different vocabulary that is worth following.

We can prove a corollary that is important to understand.

For a set of formulas Γ (Γ can be infinite), we say Γ *entails* a formula α, denoted $\Gamma \models \alpha$, if for every valuation v such that all formulas in Γ hold under v, α also holds under v.

Note that if Γ is finite, $\Gamma \models \alpha$ is equivalent to saying $(\bigwedge_{\beta \in \Gamma} \beta) \Rightarrow \alpha$ is valid. But when Γ is infinite, we can write such a formula as it would be infinite. Also note that $\Gamma \models \alpha$ iff $\Gamma \cup \{\neg\alpha\}$ is not satisfiable.

We can now show:

Corollary 2.1 *Let Γ be a set of formulas and $\Gamma \models \alpha$. Then there is a finite subset $S \subseteq \Gamma$ such that $S \models \alpha$.*

Proof Let $\Gamma \models \alpha$. Then $\Gamma \cup \{\neg\alpha\}$ is not satisfiable. By compactness theorem, there is a finite subset S' of $\Gamma \cup \{\neg\alpha\}$ that is not satisfiable. Let $S = S' \setminus \{\neg\alpha\}$. Then $S \subseteq \Gamma$ and $S \cup \{\neg\alpha\}$ is not satisfiable. But then $S \models \alpha$ holds.[1] □

2.4 Resolution

The material in this section is essentially taken from Uwe Schöning's book "Logic for Computer Scientists", Chapter 1, Section 1.5. Mild modifications were done. We refer the reader to the above source for a more elaborate introduction to resolution.

Resolution is a mechanical procedure to prove *unsatisfiability* of propositional formulas. To prove a propositional formula α valid, we can take $\neg\alpha$ and prove it to be unsatisfiable using resolution. Hence resolution is a mechanical procedure for validity as well. Instead of traditional proof systems that work directly in showing validity, resolution works by *refutation*— showing that the negation of the formula is not satisfiable. As we will see, resolution is also a *complete* procedure– if a formula is unsatisfiable, there is a proof of its unsatisfiability given as a resolution proof.

Given a formula α that we want to show unsatisfiable, we first convert it to CNF form. We skip details of how to convert to CNF form— see a standard textbook for this. Note that in practice, it's best to convert α to an equisatisfiable (not necesarily equivalent) formula α' in CNF form so that there is only a *polynomial* size blow-up (this is called Tseitin's construction; again look this up).

We now assume that α is in CNF form, i.e., of the form

$$C_1 \wedge C_2 \wedge C_m$$

where each C_i is of the form

$$L_1 \vee L_2 \vee L_{r_i}$$

where each L_j is a literal– i.e., either a proposition (p) or a negation of a proposition ($\neg p$).

It will be convenient to express CNF formulas as a set of sets of literals. A set of sets of literals of the form

[1] Note that this proof works even when $\neg\alpha$ is not in S'

$$\{\{p, \neg q, r\}, \{r, s, \neg p\}\}$$

represents the formula

$$(p \vee \neg q \vee r) \wedge (r \vee s \vee \neg p)$$

. It should be clear that any formula in CNF can be represented this way, and any such set of sets of literals corresponds to a CNF formula. Note that the order or multiplicity of elements in the sets do not matter (as they are sets) and this makes sense because disjunctions and conjunctions are associative and commutative.

More generally, we represent a formula in CNF as a set of sets of literals:

$$(L_{11} \vee \ldots L_{1,n_1}) \wedge (L_{21} \vee \ldots L_{2,n_2}) \wedge (L_{k1} \vee \ldots L_{1,n_k})$$

The sets of literals that make up the outer set are called *clauses*. For instance $\{p, \neg q, r\}$ is a clause in the example given earlier.

For a literal l, let $\bar{l}$ denote the negation of l obtained by negating l if l is a proposition, or removing the negation from it if it's a negated proposition. In other words, $\overline{p} = \neg p$ and $\overline{\neg p} = p$.

Definition 2.1 (Resolvent) Let C and C' be two clauses. The clause R is said to be a resolvent of C and C' if there is a literal l such that $l \in C$, $\bar{l} \in C'$ and

$$R = (C \setminus \{l\}) \cup (C' \setminus \{\bar{l}\}$$

Note that two clauses can have multiple resolvents (depending on which literal you choose to resolve with respect to). Also, note that R can be empty. The empty clause is an empty disjunction, which is equivalent to *false*. (Think of a clause as saying that one of the literals in it must be true; then if the clause is empty, then the clause can never be satisfied; hence it is equivalent to *false*. We denote the empty clause by $\square$.

Lemma 2.4 (Resolution Lemma) *Let F be a set of clauses. Let R be a resolvent of two clauses in F (need not be* different *clauses). Then F is equivalent to $F \cup \{R\}$.*

Proof Let v be a valuation. If $v \models F \cup R$, then clearly $v \models F$.

Now assume that $v \models F$. Let R be obtained by resolving C and C' with respect to a literal l, where $l \in C$ and $\bar{l} \in C'$. We have to argue that v satisfies some literal in R. Now either $v \models l$ or $v \models \neg l$. If $v \models l$, then since v makes C' true, it must make some literal in $C' \setminus \{\bar{l}\}$ true (as it cannot make $\bar{l}$ true), and hence makes R true. Similarly, if $v \models \bar{l}$, then since v makes C true, it must make some literal in $C \setminus \{l\}$ true, and hence makes R true. $\square$

Definition 2.2 Let F be a set of clauses. Then

$$Res(F) = F \cup \{R \mid R \textit{ is a resolvent of two clauses in } F$$

Also,

$$Res^0(F) = F$$
$$Res^{n+1}(F) = Res(Res^n(F))$$
$$Res^*(F) = \bigcup_{n \in \mathbb{N}} Res^n(F)$$

Theorem 2.2 (Resolution Theorem for Propositional Logic (finite)) *A finite set of clauses F is unsatisfiable iff $\square \in Res^*(F)$.*

Proof Let $\square \in Res^*(F)$. Then $\square \in Res^n(F)$ for some n. However, note by Resolution Lemma that adding resolvents keep the formula equivalent. Hence F is equivalence to $Res_1, Res_2, \dots Res_n$. But clearly Res_n is unsatisfiable as it contains the empty clause $\square$. So F is unsatisfiable.

The other direction is much more involved. Let F be unsatisfiable. We have to show that $\square$ is in Res^*, i.e., $\square$ is in Res_i, for some i. Let us prove this by induction on the *number of proposition symbols mentioned in F*.

Base case: When $n = 0$, F must be $\{\square\}$, but then $\square \in Res^*(F)$. Note that F cannot be $\emptyset$ as then it is satisfiable.

Induction step: Let $n > 0$ be an arbitrary positive integer. Let F be a formula over n propositions, say $\{p_1, \dots, p_n\}$. We assume the induction hypothesis, which is that for any formula G over r propositions, for any $r < n$, which is unsatisfiable, $\square \in Res^*(G)$.

Now, we first remove *trivial* clauses from F to get F'. A clause is trivial if it contains both a proposition and its negation. Note that such clauses are satisfied trivially by *all* valuations. It is easy to see that F is satisfiable iff F' is satisfiable. So we work with F'.

We now construct two subsets of F'. The first subset F_0 is obtained by taking only clauses in F' that do not contain $\neg p_n$, and then removing p_n from each of those clauses (if it exists). The second subset F_1 is similarly obtained by taking only clauses in F' that do not contain p_n, and then removing $\neg p_n$ from each of those clauses (if it exists).

(Note: There could be clauses that neither have p_n or $\neg p_n$. These clauses are unaffected and are included in both F_0 and F_1).

Intuitively, F_0 depicts the formula where we have made the decision to set p_n to false. Any clause that have $\neg p_n$ is satisfied, and hence is removed. The clauses that have p_n can be satisfied by only satisfying one of the other literals in the clause; hence we remove p_n from these clauses. Similarly, F_1 represents the formula obtained after making the decision to set p_n to true.

In fact, F_0 and F_1 both must be unsatisfiable. If F_0 were satisfiable, we can extend a satisfying valuation for F_0 by mapping p_n to false, and satisfy F. Similarly, F_1 is also unsatisfiable.

Since F_0 and F_1 are formulas that do not mention p_n, they mention strictly less than n propositions, and hence we can apply the induction hypothesis. Hence $Res^*(F_0)$ and $Res^*(F_1)$ both contain $\square$.

This means there is a sequence of clauses $C_0, \dots, C_m$ such that each C_i is in F_0 or is a resolvent of clauses C_a and C_b, with $a, b < i$. We can add the literal p_n back to all the clauses that had them to obtain a sequence of clauses resolved using clauses in F. This will result in a final clause that is either $\square$ or $\{p_n\}$.

There is a similar sequence of clauses for F_1, and adding $\neg p_n$ back to the clauses where we had removed them results again in a sequence of clauses resolved from F that yields a final clause that is either $\square$ or $\{\neg p_n\}$.

If either of the above sequences yielded $\square$, we are done, as then $\square$ does belong to $Res^*(F)$. If the sequences yielded, p_n and $\neg p_n$, then we can stitch the two sequences one after the other, and then resolve the clauses $\{p_n\}$ and $\{\neg p_n\}$ to obtain $\square$, showing $\square \in Res^*(F)$. $\square$

The above suggests the following resolution proof procedure. Let $res(F, C, C', p)$, where $C, C' \in F$, $p \in C$, $\neg p \in C$ denote $F \cup \{R\}$, where r is obtained by resolving C and C' with respect to p (i.e., $R = (C \setminus \{p\}) \cup (C' \setminus \{\neg p\})$).

Input: a formula F in CNF
Algorithm:

```
1. while ($\square \notin F$ and there is $C, C' \in F$, $p \in \mathcal{P}$, $p \in C$, $\neg p \in C'$ and
   $Res(F, C, C', p) \notin F$  {
2.     Pick such a $C, C', p$
3.     $F := Res(F, C, C', p)$;
4. }
5. If $\square \in F$ then return "unsat" else return "sat"
```

The above procedure nondeterministically picks a pair of clauses that can be resolved to get a new clause, and adds this new clause to the current formula, till either $\square$ is derived or the set stabilizes. Note that there are "only" 4^n clauses, and hence the algorithm must terminate eventually.

We can also prove an analog of the resolution theorem for *infinite* sets of clauses.

Theorem 2.3 (Resolution Theorem for Propositional Logic (infinite)) *A set of clauses F is unsatisfiable iff* $\square \in Res^*(F)$.

Proof Showing that if $\square \in Res^*(F)$, then F is unsatisfiable is easy, similar to the earlier theorem.

Now, assume that F is unsatisfiable. By compactness theorem, we know that there is a finite subset $G \subseteq F$ that is already unsatisfiable. Consequently, using the previous theorem, $\square \in Res^*(G)$. But $Res^*(G) \subseteq Res^*(F)$. So $\square \in Res^*(F)$. $\square$

Chapter 3
Quantifier Elimination and Decidability

3.1 Quantifiier Elimination

Fix a signature and the first order logic L over this signature. Let C be a nonempty class of structures. The class could consist of just a single structure, like $(\mathbb{R}, <)$ or a class of structures, like all dense linear orders without a max/min element.

We say that L over C admits *quantifier elimination* if for every formula $\varphi(\mathbf{x})$, there is a quantifier free formula $\varphi^*(\mathbf{x})$ such that φ and φ^* are equivalent over all structures in C, i.e., for every model $M \in C$ and every interpretation s, $M, s \models \varphi$ iff $M, s \models \varphi^*$.

Note that we do not say above that φ^* be *computable* from φ, but this will often be the case when L admits quantifier elimination. Notice that we require the above for *formulas*, not just sentences. A formula $\varphi(\mathbf{x})$ can be seen as a property that identifies a subset of vectors of elements in a model, and the above demands that the quantifier-free formula φ^* capture the same subset of vectors.

For example, consider the formula $\exists x.\ a \cdot x \cdot x + b \cdot x + c = 0$, over reals with addition and multiplication, where a, b, c, x are variables. Then the above formula identifies the triples $\langle a, b, c \rangle$ such that a (non-imaginary) real root of the equation $ax^2 + bx + c$ exists. Then an equivalent quantifier-free formula that identifies the same set of triples would be $b \cdot b - 4 \cdot a \cdot c > 0$ (which we know from school algebra on roots of a quadratic equation).

If L over a class C admits (effective) quantifier elimination and if we can decide formulas quantifier-free formulas without any variables (i.e., terms are built only from constants), then this leads to a decidability procedure for sentences. Given a sentence, we can find the equivalent quantifier free formula (which won't have free variables, and hence have no variables) equivalent to it, and check to see if it is true over the class of models. Since it is often true that quantifier elimination is effective and that deciding sentences without quantifiers is often decidable, it is often the case that a logic L over a class C admits quantifier instantiation implies decidability of validity of the logic L over the class of structures C.

Eliminating one quantifier

It is easy to see that if for every formula of the form $\exists x.\varphi(x, \mathbf{y})$, where φ is *quantifier-free*, there is a quantifier-free formula $\varphi^*(\mathbf{y})$ that is equivalent to it over all structures in C, then L over C admits quantifier instantiation. The reason for this is that we can take any formula, express $\forall$ quantification using $\exists$ and negation, and starting with the innermost quantified formulas, systematically eliminate one quantifier at a time in order to eliminate all quantifiers. Hence eliminating formulas with a single existential quantifier is all that is needed to show that a logic admits quantifier elimination.

Dealing only with conjunctions of literals

We can make one more simplifying assumption, if convenient. We can assume that when doing quantifier elimination of a formula $\exists x.\varphi$, φ is a conjunction of literals, i.e., a conjunction of atomic formulas or their negation. The reason is that for any arbitrary formula $\exists x.\psi$, where ψ is quantifier-free, we can always treat it as a Boolean formula over atomic formulas, and write it in disjunctive normal form. Then, we notice that $\exists$ quantifier distributes over disjunctions, and hence we can write the formula as a disjunction of formulae of the form $\exists x.C$, where C is a conjunction of literals. If we can do quantifier elimination on such formulae, we can simply do this for each disjunct to obtain a quantifier-free formula.

3.2 Dense Linear Orders without Endpoints

In this section, we will show that FOL over the particular model $(\mathbb{R}, <, =)$ admits quantifier elimination.

When doing the proof that it admits quantifier elimination, we will use the following properties about this particular model, which themselves can be formulated in FOL (we call these the DLOWE axioms):

- $<$ is a strict partial order:
  $<$ is irreflexive:
  $$\forall x, y : \neg(x < x)$$
  $<$ is asymmetric:
  $$\forall x, y : (x < y) \implies \neg(y < x)$$
  $<$ is transitive:
  $$\forall x, y, z : ((x < y) \wedge (y < z)) \Rightarrow (x < z)$$
- $<$ is total:
  $$\forall x, y : (x < y \vee x = y \vee y < x)$$
- The ordering is dense:

$$\forall x, y.(x < y \Rightarrow \exists z.(x < z \land z < y))$$

- There are no minimum or maximum elements (no endpoints):

$$\forall x\ \exists y.\ x < y$$

$$\forall x\ \exists y.\ y < x$$

It turns out that when proving that the reals with < admits quantifier elimination, we will use *only* the above properties about $(\mathbb{R}, <)$. Consequently, it turns out that the decision procedure we build is a decision procedure for *any* structure that satisfies the above axioms. Since this decision procedure clearly defines a set of sentences (those that it says are the only ones that are true in the model $(\mathbb{R}, <)$ and hence the others are false), it follows that any structure that satisfies the above axioms must satisfy the *same* set of sentences that $(\mathbb{R}, <)$ satisfies. For instance, the structure $(\mathbb{Q}, <)$ satisfies all the axioms above and hence has the same theory as reals with <.

Consequently, the theory of any structure with a < relation that satisfies the above axioms is fixed and is (negation) complete— i.e., any sentence either holds in all the structures or holds in none of them. No first-order logic formula (over this signature involving < and = only) can distinguish between any two structures that satisfy the axioms (like rationals and reals).

Two structures that have the same first-order theory are called *elementary equivalent*. Once we show quantifier elimination, we would have shown that $(\mathbb{Q}, <)$ and $(\mathbb{R}, <)$ are elementary equivalent.

In general, for any infinite structure S, it will always be the case that there is a different (non-isomorphic) structure S' that will have the same theory as S (we will prove this later). So it is good to start understanding that FO cannot distinguish between structures well enough and there are always alternate structures that satisfy the same properties that one structure satisfies.

Quantifier Elimination for DLOWE

We briefly give the steps for eliminating the quantifier in a formula of the form $\exists x.\varphi(x, \mathbf{y})$, where φ is quantifier-free, and is a conjunction of literals.

First, note that since there are no functions, atomic formulas are of the form $t < t'$ and $t = t'$ only, where t, t' are variables. We can get rid of negations, by rewriting negated atomic formulae:

$$\neg(t < t') \equiv (t = t' \lor t' < t)$$

$$\neg(t = t') \equiv (t < t' \lor t' < t)$$

We can do this first, before converting to DNF, etc. Hence we can assume φ is a conjunction of atomic formulae only.

We can also assume, without loss of generality, that all the atomic formulae in φ involve x. After all, if an atomic formula does not involve x, it can be "pulled" out of the quantification as a separation conjunct, for example:

$$\exists x.(y_1 < y_2 \wedge \psi) \equiv y_1 < y_2 \wedge \exists x.\psi$$

Also, if there is a conjunct of the form $x < x$, then clearly the formula is equivalent to *false*, and we are done eliminating quantifiers (we can write *false* as say $0 < 0$).

So let us assume φ is a set of conjunctions of atomic formulae, each being of the form $x = t$, $x < t$, and $x > t$.

The first case is when φ has a conjunct of the form $x = t$. In this case, we can clearly substitute x with t, as it is the only way to satisfy the formula, and we are done.

$$\exists x.\ (x = t \wedge \varphi(x, \overline{y})) \quad \equiv \quad \varphi[t/x](\overline{y})$$

The remaining case is when there are no conjuncts of the form $x = t$ in φ, and hence $varphi$ consists only of atomic formulas of the kind $t < x$ and $x < t$. Let L be the set of terms t such that $t < x$ occurs in the formula and U be the set of terms such that $x < u$ occurs in the formula. Then there is some x satisfying the constraints iff every term in L is less than every term in U. Clearly this condition is necessary as the formula demands it. It is also sufficient: When L and U are nonempty, it is also sufficient since if the condition is satisfied, we can pick x between the "largest" element in L and the "smallest" element in U, since the order is dense. Note that if L is empty and U is nonempty, then the condition doesn't demand anything (i.e., it is *true*, which can be written as $0 = 0$). And this condition is sufficient too as we can pick x to be something smaller than all elements in U, as there is no minimal element. Similarly, the condition is sufficient when L is nonempty and U is empty. We can summarize the elimination as:

$$\exists x.(\bigwedge_{t \in L} (t < x) \ \wedge \ \bigwedge_{t' \in U} (x < t') \ ) \quad \equiv \quad \bigwedge_{t \in L, t' \in U} (t < t')$$

with the understanding that the empty set of conjuncts is written as $0 = 0$ (synonymous for *true*).

The above finishes the proof that the theory of $(\mathbb{R}, 0, <, =)$ admits quantifier elimination. Since the quantifier elimination is *constructive*, and since quantifier-free variable-free (grounded) formulas can be checked (they are just Boolean combinations of atomic formulas of the kind $0 < 0$ and $0 = 0$)), it follows that checking whether a formula is valid is decidable. Since we followed only the axioms of DLOWE, it follows that the same procedure decides the theory of all linear orders without end-points.

Theorem 3.1 *The first-order logic over* $(\mathbb{R}, 0, <, =)$ *admits quantifier elimination. The first-order logic over dense linear orders admits quantifier elimination using the same elimination procedure. Their theories are hence decidable and identical. The theory of* $(\mathbb{Q}, 0, <, =)$ *is hence also identical and decidable.*

The complexity of the procedure is quite high. Eliminating one quantifier causes a quadratic blowup in the formula. However, when m quantifiers are eliminated, this results in a formula possibly of size $O(n^{2^m})$, which is doubly exponential in the number of quantifiers.

3.3 Quantifier Elimination for rationals with addition: $(\mathbb{Q}, 0, 1, +, -, <, =)$

See book Calculus of Computation, Section 7.3.

The technique is by Ferrante and Rackoff. The technique does not require to put the quantified formula in disjunctive normal form, but works directly on Boolean combinations.
Briefly:

Step 1. Given $\exists x.\varphi(x, \overline{y})$, put it in negation normal form (NNF), where φ is expressed as a postive Boolean combination of atomic formulae and negations of atomic formulae. Note that atomic formulae are of the form $s < t$ or $s = t$, where s, t are terms involving constants, variables, and combinations using +, i.e., they are linear expressions.
Step 2. Replace literals with equivalent formulae that do not have negation:

$$\neg(s < t) \equiv (s = t \vee s > t)$$

$$\neg(s = t) \equiv (s < t \vee t < s)$$

Step 3. "Solve for x" in each term. Write each atomic formula in the form $x < a$, $b < x$, or $x = c$. You may have to use division by constants (this will be temporary and get removed in the end) to do this.

Key Idea: We now come to the key idea. Think of the terms involved in the various atomic formulas above (of the kind a, b, c) above. Let S be the set of such terms (they don't involve x). Depending on the valuation of the free variables $\overline{y}$, the terms will be some rational numbers; think of them on the rational "number line". Now, x can be any rational number, but if two terms t_1 and t_2 evaluate to two "consecutive" values on the number line, it doesn't matter which value of x we pick in between t_1 and t_2... all of them will make the atomic formulas evaluate the same way. So we can just pick $(t_1 + t_2)/2$. Now, we don't know what *order* the terms will evaluate. But we can simply instantiate x to $(t + t')/2$ for *every pair* of terms t, t'. (We do this also for the pair of terms t, t as well— which will just instantiate x by t; this will cover x being precisely equal to one of the terms).

To cover the range of rationals less than *all* the terms, we can instantiate x to something smaller than all the terms. But instead of doing this as an instantiation, we just imagine instantiating x to some large negative value, and see how atomic formulas will evaluate. Clearly, atomic formulas of the form $x < a$ will evaluate to *true*, $x = t$ will evaluate to *false*, and $x > t$ will evaluate to *false*. So we can replace the atomic formulas by these values, and get a formula $F_{-\infty}$.

Similarly, to cover the range of rationals larger than all the terms, we pretend instantiating x to a value much larger than the values of all the terms. The atomic formulas $x > t$ evaluate to *true*, and the formulas of the form $x = t$ and $x < t$ evaluate to *false*. Replacing these gives the formula $F_{+\infty}$.

We then take the disjunction of all the above formulas to eliminate x.

Step 4. Let the current formula be $\exists x.F_3$ Let S be the set of terms not involving x in the formula.

Let $F_{-\infty}$ be the formula by replacing each atomic formula of the form $x < a$ by *true*, and the atomic formulae of the form $x = b$ and $x > c$ by *false*.

Let $F_{+\infty}$ be the formula by replacing each atomic formula of the form $x > c$ by *true*, and the atomic formulae of the form $x = b$ and $x < a$ by *false*.

The quantifier-eliminated formula is now:

$$F_{-\infty} \vee F_{+\infty} \vee \bigvee_{s,t \in S} F_3[(s+t)/2 \;/\; x]$$

The above formula is quantifier-free and equivalent to the original formula on $\mathbb{Q}$. You can now rewrite the above by multpiplying by constants to use only integer constants, and integers can be written using an arithmetic expressions over 0 and 1.

A note of axiomatization of rationals with addition::

One can ask, similar to dense linear orders without endpoints, whether we can formulate a set of axioms that capture the properties of rationals with addition that we are utilizing in order to do quantifier elimination. One can indeed formulate such a set of axioms, and this set of axioms would hence capture the theoru of rationals with addition completely. However, this needs to be carefully done.

The book Calculus of Computation referred to here does quantifier elimination and, in parallel, gives a set of axioms for rationals with addition. *However, the book does not actually prove the axioms are precisely the properties that are used when doing quantifier elimination.* Hence it does not in any way show that the axioms presented are complete. (In fact, the axioms presented are clearly not complete as they do not capture any property of the constant 1). The book, unfortulately, seems to suggest that the axiom system and the quantifier elimination are linked, and hence that the axioms are complete. While one can see how this can be done (for a mildly modified axiom system), such an argument or proof is not presented in the book.

Example 3.1 Let's illustrate with a simple example.

Consider the formula:

$$\varphi : \forall y.\ (0 < x \Rightarrow 1 < x + y)$$

The above formula is a formula with a single free variable y, and it is easy to see that the formula holds for a particular interpretation of y iff $y \geq 1$. Hence we expect our quantifier eliminated formula to be equivalent to $y \geq 1$.

The formula can be written as:

$$\neg\exists\neg(0 < x \Rightarrow 1 < x + y)$$

Let's take the inner formula

$$\psi : \exists x.\ \neg(0 < x \Rightarrow 1 < x + y)$$

and eliminate the quantifier.

Pushing negations in, we get the equivalent formula

$$\exists x.\ (0 < x \wedge \neg(1 < x + y))$$

which is equivalent to

$$\exists x.\ (0 < x \wedge (1 = x + y \vee x + y < 1))$$

Writing it in a form where x is grouped alone, and coefficient 1, on one side of every term,

$$\exists x.\ (0 < x \wedge (x = 1 - y \vee x < 1 - y))$$

The $F_{-\infty}$ formula is $(\mathit{false} \wedge (\mathit{false} \vee \mathit{true}))$ which is (Boolean) equivalent to *false*.

The $F_{+\infty}$ formula is $(\mathit{true} \wedge (\mathit{false} \vee \mathit{false}))$ which is (Boolean) equivalent to false.

The terms in the formula that do not involve x are $S = \{0, 1 - y\}$ So the formula is equivalent to

$$\bigvee_{t,t' \in S} F_3[(t + t')/2 \,//\, x]$$

which has the following disjuncts:

$$0 < 0 \wedge (0 = 1{-}y \vee 0 < 1{-}y)$$

$$0 < 1{-}y \wedge (1{-}y = 1{-}y \vee 1{-}y < 1{-}y)$$

$$0 < (0 + 1 - y)/2 \wedge ((0 + 1 - y)/2 = 1 - y \vee (0 + 1 - y)/2 < 1 - y)$$

Writing them by removing division by constants gives the disjuncts:

$$0 < 0 \wedge (0 = 1{-}y \vee 0 < 1{-}y)$$

$$0 < 1{-}y \wedge (1{-}y = 1{-}y \vee 1{-}y < 1{-}y)$$

$$0 + 0 < 0 + 1 - y \wedge (0 + 1 - y = 1 - y + 1 - y \vee 0 + 1 - y < 1 - y + 1 - y)$$

We shouldn't simplify further (in fact, we should include $F_{-\infty}$ and $F_{+\infty}$ as well. But notice that the first disjunct is equivalent to *false*, the second formula is equivalent to

$y < 1$ and the third formula is equivalent to $y < 1 \wedge (y = 1 \vee y < 1$, i.e., equivalent to $y \leq 1$.

Going back to the original formula φ, we need to negate the above formula which gives the negation of the above fomulas as conjuncts:

$$\neg(0 < 0 \wedge (0 = 1{-}y \vee 0 < 1{-}y))$$

$$\neg(0 < 1{-}y \wedge (1{-}y = 1{-}y \vee 1{-}y < 1{-}y))$$

$$\neg(0 + 0 < 0 + 1 - y \wedge (0 + 1 - y = 1 - y + 1 - y \vee 0 + 1 - y < 1 - y + 1 - y))$$

The above is the quantifier-eliminated formula equivalent to φ.

These conjuncts are semantically equivalent to *true*, $y \geq 1$ and $y > 1$. Hence the conjunction is semantically equivalent to $y > 1$, which is what we expect.

3.4 The Theory of Reals with Addition

Let us consider the theory of reals with addition, i.e., the theory of the model $(\mathbb{R}, 0, 1, +, -, <=)$. Now notice that when arguing why the quantifier elimination procedure we described for rationals with addition above, we used properties that are equally true for reals! Rewriting negated atomic formulas, the key idea of choosing the average of every pair of terms and beyond the minimum and maximum evaluated terms ($F_{-\infty}$ and $F_{+\infty}$), etc., are valid for reals too. Consequently, the quantifier-elimination based decision procedure works equally well for the sentences in the theory of reals with addition.

Now, if the same decision procedure works for deciding two theories, then surely the theories must be the same! Hence $Th(\mathbb{R}, 0, 1, +, -, <, =) = Th(\mathbb{Q}, 0, 1, +, -, <, =)$. Note that this means that there is no first order sentence that can distinguish between the two structures. If we had multiplication in our signature, the formula $\exists x.(x \times x = 1 + 1)$ would be true over reals but not over rationals. However, with only addition, reals and rationals are not distinguishable.

3.4.1 Aside: Axiomatizations

Again, we can ask whether the properties we used to do quantifier elimination for rationals/reals with addition can be codified as a set of first order axioms. Again, this is largely possible. The book Calculus of Computation gives such an axiomatization (see Chapter 3) and suggests that the quantifier elimination procedure is for the theory of this axiomatization (as opposed to the *true* theory of rationals); but the proofs later do not really show this.

In general, it is true however that for theories of a *single* structure (or a class of structures whose theory is negation complete, i.e., for any sentence α, either α

or $\neg\alpha$ is in the theory), the notions of the existence of a recursive axiomatization and decidability are synonymous. One direction is easy— if the theory is decidable, we can simply take the theory itself as its recursive axiomatization (sounds like we are cheating, but we are not). And if there is a recursive axiomatization for a complete theory, it will follow, as we will show later (Gödel's strong completeness theorem), that the membership problem is recursively enumerable, and hence by simultaneously checking if α or $\neg\alpha$ is in the theory, we can show the problem is decidable.

3.4.2 Other theories that admit quantifier elimination

There are several other important theories that admit quantifier elimination.

First, the theory of integers with addition, often referred to as Presburger arithmetic, is decidable; this theory is defined by the model $(\mathbb{Z}, 0, 1, +.-, < . =)$. However, the first-order logic over this theory does not, by itself, admit quantifier elimination. For example, one can show that there is no quantifier-free formula that is equivalent to the formula $\exists x.x + x = y$, which says y is even. However, we can *extend* the language so that it admits quantifier elimination. We introduce predicates of the form $c|t$, where c is a constant, to the logic. Then this extended logic does admit quantifier elimination, and leads to a decision procedure. See Calculus of Computation.

Another important theory that admits quantifier elimination is the theory of reals with addition and multiplication: $(\mathbb{R}, 0, 1, +, \cdot, \leq, =)$. And is decidable! This theorem is basically due to Tarski, and is called *Tarski-Seidenberg theorem*.

Chapter 4
Validity of FOL is undecidable and is r.e.-hard

In this chapter, we look at general FOL (over arbitrary signatures). We will first consider the problem of checking if a formula is valid over all models, and show this to be undecidable. In particular, we will show that validity is r.e.-hard. In a later chapter, in what essentially constitutes proving Gödel's completeness theorem, we will see that validity is a problem that is r.e.. So the validity problem is of the same complexity of the halting problem. And there is a Turing machine that can print out the list of all valid formulas. In fact, we will prove a stronger result that for any recursive (decidable) set of axioms A, the theorems that are semantically entailed by A is r.e. (i.e., the set of sentences true in all models that satisfy the axioms A is r.e.).

The problem of checking validity of a first-order formula is referred to sometimes as the *classical decision problem* [**?**] or the *Entscheidungsproblem*, and the problem was popularized by David Hilbert (*das Entscheidungsproblem*, or *the decision problem*), in an attempt to lead towards the formalization of mathematics. However, what *computation* meant was not clear then. These were resolved in 1936, when Church postulated that a class of computable functions using recursion captures computability, and proved that the classical decision problem was not solvable using this notion of computability. A few months later, Alan Turing, in his paper that introduced Turing machines (and started the field of theoretical computer science, or even computer science), also examined the *Entscheidungsproblem* (mentioned in the title of the paper), and showed validity of first-order logic is undecidable. Soon people realized that the notions of computing defined by Turing and Church were the same (Turing in fact showed equivalence in his paper), and the Church-Turing postulate was that the notion of computability coincided with the notion of computability defined by λ-calculus and Turing machines. The undecidability of the *Entscheidungsproblem* is credited now to both Church and Turing.

In this section, we will also look at the class of all finite models $\mathcal{F}$, and show that validity of formulas over all finite models is also undecidable. The reason we do this proof now is that the proof is very similar, but with important differences. In particular, we will show that over finite models, validity is co-r.e. hard, and hence not r.e.. Consequently, there is no proof system that can prove all properties about all finite models! This fundamental incompleteness result is easier to understand than the incompleteness result for arithmetic which we will see later (i.e., Gödel's first incompleteness theorem).

A note on functions and relations

In general, we could ask why we need both functions and relations in FOL. We could for example encode functions as relations. Given a function symbol f, we could introduce in its stead a relation symbol R_f, and demand that:

(a) for every x, there is a y such that $R_f(x, y)$ holds, i.e.,

$$\forall x.\ \exists y.\ R_f(x, y)$$

(b) for every x, y, and z, if $R_f(x, y)$ and $R_f(x, z)$ hold, then $y = z$, i.e.,

$$\forall x, y, z.((R_f(x, y) \wedge R_f(x, z)) \Rightarrow y = z)$$

These two properties, formulated in FOL over the signature with R_f instead of f, captures the property that R_f encodes a function. Furthermore, we can take any formula involving f and translate it to one involving R_f — this would involve quantifying over a variable for each term of the form $f(t)$ in the formula, and then relating t and this variable using R_f, etc., but it is possible. Consequently, many of our results will not depend on the fact that the signature includes functions.

However, there are some cases where logics with relations are simpler to reason with than logics with relations and functions. One case is when dealing with the logic fragment that has *only* universal quantification. For instance, for this fragment, the validity problem when there are only relation symbols is *decidable*, while the validity problem in the presence of relation and function symbols (or only function symbols) is *undecidable*. Note that there is no contradiction here as one of the formulas above that capture R_f is a $\forall\exists$ formula.

4.1 Validity of FOL is r.e.-hard

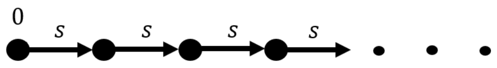
Let us consider the problem of checking if a given sentence φ over a first-order signature is valid, i.e., whether it is true in all models. For example, the sentence $(\forall x.(p(x) \Rightarrow q(x)) \wedge p(c)) \Rightarrow q(c)$ is a valid formula, where c is a constant symbol, and p, q are unary predicates.

One may wonder first whether a sentence always has a finite model, if at all it has a model. If this were true, then we could negate the sentence and try to find a finite model for it, say by enumerating all finite models systematically, and perhaps hope that there will be a bound. But it is easy to see that this is *not* true. In fact, as we will see later, it is *good news* that finite models do not always exist, as when restricted to finite models, validity is not even r.e.!

Consider the following formula involving a function symbol s and a constant symbol 0.

$$(\forall x.\neg(s(x) = 0)) \wedge (\forall x, y.(s(x) = s(y) \Rightarrow x = y))$$

The above says that the successor (s) of no element is 0, and that no element is the successor of two different elements. In any model where this formula holds, it is easy to see that $s(0)$ must be different from 0 (since the successor of no element can be 0), $s(s(0))$ must be different from 0 (since the successor of no element can be 0 and there can be only one element whose successor is $s(0)$). One can argue by induction that for any $i \in \mathbb{N}$, $0, s(0), \ldots, s^i(0)$ must all be distinct elements. Consequently, any satisfying model has to be infinite.



We will use the "*number line*" created by the above formula crucially in our proof, which will reduce the halting problem of Turing machines. However, we modify it slightly so that we use a relation instead of a function.

Let s be a binary relation. Then we will, use the following three properties to encode the infinite number line:

$$\begin{aligned}
&\textit{NL1}: \ \ \forall x.\ \exists y.s(x, y)\\
&\textit{NL2}: \ \ \forall x.\ \forall y.\ \forall z.\ ((s(x, y) \wedge s(x, z)) \Rightarrow y = z)\\
&\textit{NL3}: \ \ \forall x.\ \neg(s(x, 0))\\
&\textit{NL4}: \ \ \forall x.\ \forall y.\ \forall z.\ ((s(x, z) \wedge s(y, z)) \Rightarrow x = y)
\end{aligned}$$

The first two formulas demand that the relation s encodes a function. The second two encode the requirement that ensure the element 0, the element x_1 such that $s(0, x_1)$ holds, the element x_2 such that $s(x_1, x2)$ holds, etc., are all different from each other.

Aside: If we drop the requirement (NL1) above, then we get s is a partial *function, and we can see that the other conditions will still ensure distinctness of elements successively related to* 0*. However, we could get a* finite *prefix of the number line (i.e., we may not get infinitely many elements). We will use this for the second proof showing that validity over finite models is undecidable as well. For the current proof, however, it is important that the number line is infinite.*

Proof of undecidability:

We will show that the halting problem for Turing machines reduces to the validity problem for FOL. We will use the constants 0 and 1, and insist that 1 is the successor of 0 using the following formula φ_0:

$$s(0,1)$$

Since we are going to only worry about whether a Turing machine halts on empty input, let us denote a deterministic Turing machine as $M = (Q, \Delta, \delta, q_0, q_h)$. Q is a finite set of states, with $q_0 \in Q$ being the initial state, and q_h be the halting state. Γ is the finite tape alphabet, and the transition relation $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$. Let us also assume that all transitions from q_h go to state q_h, without loss of generality.

Let us assume that the states Q are $\{0, 1, \dots, m\}$ for some $m \in \mathbb{N}$, $m > 0$, and assume the initial state is 0 and the halting state is 1. Let us also assume that the tape symbols Δ are $\{0, \dots, n\}$, for some $n \in \mathbb{N}$, $n > 0$. We assume 0 stands for the blank symbol, and we assume 1 is a special symbol used to mark the left end of the tape–the Turing machine can be assumed to have no rewrites of this symbol nor move left when it reads it so that it never goes off the left end.

The Turing machine's computation can be seen as an infinite (and unique) sequence of configurations $C_1, C_2, \dots, C_t, \dots$ where C_t is the configuration at time t. Each configuration includes the content of the tape, the position of the head on the tape, and the state the machine is in. We assume that the Turing machine from the halting state, stays in the halting state, and hence this infinite sequence of configurations always exists and is unique.

The crucial idea now is to encode the Turing machine's computation as a model. In this model, we will have the constant 0, and the s-related successory encode (at least) the number line. So, intuitively, we have elements that encode all natural numbers. We have two relations (over natural numbers):

- $T(t, c, \gamma)$ that intuitively holds true iff at the t'th configuration in the computation, the c'th cell contains the tape symbol $\gamma \in \Gamma$
- $H(t, c, q)$ that intuitively holds true iff at the t'th configuration in the computation, the head is pointing to cell c and the TM is in state $q \in Q$

Let $IsState(q)$ denote the formula that checks whether the number q represents a state, i.e., is 0 or one of the $m - 1$ successors of 0.

$$IsState(q) = (q = 0) \vee \exists x_1.(s(0, x) \wedge (x_1 = s \vee \exists x_2.\ (s(x1, x2) \wedge (x_2 = s \vee \dots \wedge (x_{m-1} = s)))))$$

Similarly, let $IsSymb(\gamma)$ denote the formula that checks whether γ represents a tape symbol, i.e., is 0 or one of the $n - 1$ successors of 0.

We hence will insist the conjunction of the following formulas (call this φ_1):

$$\forall t, c.\ \exists \gamma.\ (IsSymb(\gamma) \wedge T(t, c, \gamma))$$

$$\forall t, c, \gamma, \gamma'.(T(t, c, \gamma) \wedge T(t, c, \gamma')) \Rightarrow \gamma = \gamma'$$

$$\forall t, \exists c.\ \exists q.\ (IsState(q) \wedge H(t, c, q))$$

$$\forall t, c, c', q, q'.\ (H(t, c, q) \wedge H(t, c', q')) \Rightarrow (c = c' \wedge q = q')$$

The above formulas ensure that the relations T and H ensure that there is one and only one symbol in any cell in any configuration at any time, and that at any time, in the configuration at that time, the head points to some cell and only one cell, and the TM is in some state and only one state.

We now have to demand that the sequences that T and H encode correspond to true computations of the Turing machine. We do this by fixing the initial configuration and restricting how the configurations progress.

- The first configuration is correct (call this formula φ_2):

$$H(0,1,0) \wedge T(0,0,1) \wedge \forall x.(x \neq 0 \Rightarrow T(0,x,0)$$

  The above says that the TM is in state 0 at time 0, with head pointing to cell 1. And that in the first configuration, the first cell has the tape-symbol 1 (marking the beginning of the tape), and the rest of the cells have blank symbols (0).
- The transitions of configurations are correct. Call this formula φ_3.
  For each state q and each symbol γ, where $\delta(q,\gamma)$ is of the form (q',γ',R) (i.e., instructs the head to go right), we add the following conjunct:

$$\forall t,t',c.\ (s(t,t') \wedge s(c,c') \wedge T(t,c,\gamma) \wedge H(t,c,q)) \Rightarrow$$
$$\left(T(t',c,\gamma') \wedge H(t',c',q') \wedge \forall c''.\ \left(\neg(c=c'') \Rightarrow \bigwedge_{\gamma''\in\Gamma} T(t',c'',\gamma'') \Leftrightarrow T(t,c'',\gamma'')\right)\right)$$

  The above says that for every time t, successor time t', every cell c, and the cell to the right of it c', if the TM has symbol γ in cell c and time t with head pointing to cell c, then in the next configuration (at time t'), the cell c has symbol γ', the head points to c', the state is q', and every other cell other than c has the same content as it had in the previous configuration.

  Similarly, for each state q and symbol $\gamma \in \Gamma$ such that $\delta(q,\gamma)$ is of the form (q',γ',L), we add the following conjunct:

$$\forall t,t',c.\ (s(t,t') \wedge s(c',c) \wedge T(t,c,\gamma) \wedge H(t,c,q)) \Rightarrow$$
$$\left(T(t',c,\gamma') \wedge H(t',c',q') \wedge \forall c''.\ \left(\neg(c=c'') \Rightarrow \bigwedge_{\gamma''\in\Gamma} T(t',c'',\gamma'') \Leftrightarrow T(t,c'',s'')\right)\right)$$

  We finally have to demand that the TM halts, which the following formula demands:

$$\varphi_4 :\ \ \exists t.\ \exists c.\ H(t,c,1)$$

  Our final formula, whose validity captures the halting of the Turing machine M on the empty tape, is:

$$\psi_M :\ \ (NL1 \wedge NL2 \wedge NL3 \wedge NL4 \wedge \varphi_0 \wedge \varphi_1 \wedge \varphi_2 \wedge \varphi_3) \Rightarrow \varphi_4$$

The above formula says that if $NL1$–$NL4$ and φ_0 are true (i.e., we have 0 and infinitely many distinct successors, modeling at least the natural numbers[1], with 1 being successor of 0), and if the relations T and H encode the computation of the TM correctly (φ_1–φ_3), then we are bound to find a halting configuration at some time.

If the above formula ψ_M is valid, then to show the Turing machine must halt, consider the model where the universe is precisely $\mathbb{N}$, with symbol 0 interpreted as the number 0, and $s(n, n')$ holds iff $n' = n + 1$. Then consider the relations T and H interpreted so that they encode precisely the (infinite) fixed computation of the Turing machine starting on the blank tape. Then the antecedent of ψ_M will be satisfied by the model. Since the formula is valid, the consequent must be true, i.e., there is a finite time t and a cell c such that $H(t, c, 1)$ holds. From this we know that the TM halts by the time t.

Now for the other direction, which is easy except it involves a subtle point! Assume that M halts. Then we have to prove that the formula ψ_M is valid. Consider a model that satisfies the antecedent of ψ_M. We know that the model then must contain 0 and its successors defined by taking the s relation, as distinct elements, defining an ω-chain of elements isomorphic to $\mathbb{N}$. Let this subset of the universe by U'. Note that there can be elements that are in the universe that are not part of this chain. The only way to encode the relations T and H on the elements involving U' is to follow the Turing machine's computation. Since the Turing machine will halt, we know there will be a time $t \in U'$ and cell $c \in U'$ such that $H(t, c, 1)$ will hold. So the formula φ_4 will hold in this model, making ψ_M also hold. Hence the formula is valid.

We have now proved what is called Church's theorem:

Theorem 4.1 (Church) *The validity problem for FOL is undecidable, and is in fact r.e.-hard.*

In fact, we have shown above that validity is undecidable even if there are only three relation symbols and no function symbols. We can even strengthen the result to a single relational symbol (model the three relations using a single relation that has an additional component, which can be 0, 1, or 2, to encode the three relations in one). One can also do the proof using no constant symbols (we can just existentially quantify to get the element standing for 0). So it's hard to find any reasonable restrictions on the signature that has hope for having validity decidable. Also, if we had one function symbol, we can still do the reduction.

Corollary 4.1 *For any signature containing at least one relation symbol or one function symbol, the validity problem for FOL is undecidable, and is r.e.-hard.*

Note however that formula uses quantifiers and quantifier alternation. One could ask whether there are restricted quantifier sequences that lead to decidability. A fairly complete characterization of what is decidable and what is not is known (interested readers are referred to the book "The Classical Decision Problem" by Börger, Grädel, and Gurevich).

[1] Maybe more! See notes later.

Some subtleties of the reduction:

There is a subtle issue here worth noting. Though we have demanded that the s successors of 0 *form an infinite chain, we have not ensured that the model contains* just *this chain. There could be elements in the universe that are not "reachable" by any number of transitive relations to* 0 *(for example, we could have another infinite two-way chain). If we had access in our logic to the reflexive and transitive closure* $s*$ *of* s*, then we could have insisted* $\forall x.s*(0,x)$*, and this would have not allowed such models. However, FOL cannot express the transitive closure of a relation, and so we cannot do the above. So you should go through the above proof carefully (the second direction, in particular), and make sure it goes through even when such unreachable elements are present.*

Now, assume that we wanted to reduce the problem of whether the Turing machine does not halt *on the blank tape to FOL validity. Then the natural modification to the above proof* will not work. *We could do the same constraints as above, and demand, instead of* φ_4*, the formula* $\forall t,c.\neg(H(t,c,1))$ *to say that the TM does not halt. However, the proof will fail. Consider a TM that doesn't halt. Then we could encode the TM's computation faithfully using* T *and* H *on the infinite segment of elements reachable from* 0*. But we could have* another *element not reachable from* 0 *using* $s()$ *in finitely many steps, and have a model that maps* $H(t,c,1)$ *to be true (we may have to engineer the elements s related to such t's and c's).*

The above shows subtlety shows why it is not easy to reduce the co-r.e. hard problem of non-halting of Turing machines to the validity problem for FOL. In fact, validity is r.e., as we will prove later, and hence is not co-r.e.-hard, and such a reduction just doesn't exist.

The inability of ensuring that just 0 *and its successors are in the model also plays out when capturing the theory of arithmetic using axioms. We cannot ensure this property; in fact, we cannot even ensure this using an infinite number of axioms! Consequently, axiomatizations of number theory always include so-called "non-standard" models of arithmetic.*

4.2 Trakhtenbrot's theorem: Validity of FOL over finite models is undecidable, and co-r.e. hard

One important class of structures are *finite structures*. In computer science, especially, many of the structures that we want to reason with are finite: graphs, data-structures, relational databases, etc.

Given that the undecidability proof above crucially used infinite structures to encode Turing machines, we could ask whether the validity problem becomes easier if we consider only finite models.

The surprising and non-intuitive result here is that the validity problem becomes *harder* when we consider finite models. Having to reason with finite models is a curse, not a blessing! In fact, we are going to adapt the proof above to show that validity of FOL is not just undecidable, but is co-r.e. hard, and hence not even r.e..

Note that the *satisfiability* problem for FOL on finite structures is indeed r.e.. Since each finite model has a finite description, we can simply enumerate all finite models, and check whether any of them satisfy the given formula, and stop if we find one that does. Evaluating a formula on a finite model is also easy (though expensive!)— it just considers examining all possible valuations for variables that are universally quantified, searching for one possible valuation for variables that are existentially quantified, and evaluating the inner quantifier-free formulas.

Theorem 4.2 *The satisfiability problem for FOL over finite structures is recursively enumerable.*

So, assuming validity is undecidable for a logic, and hence so is satisfiability, we cannot have *both* problems to be r.e. (as then the problems would become decidable). It turns out that over all structures, validity is in r.e. and satisfiability is co-r.e.-hard. And it turns out that over all finite structures, validity is co-r.e.-hard while satisfiability is in r.e.. Hence if we want to prove theorems, reasoning about infinite structures is in a sense easier— at least when theorems are true, we can build machines that identify them as being true (and its computation can be thought of as a proof of the theorem). But over finite structures, even valid statements need not have proofs.

Proof of undecidability and co-r.e. hardness

Let us now prove that validity over finite models is undecidable. We will in fact reduce the *non-halting* problem for Turing machines to validity of FOL over finite models.

We will use similar ideas to the earlier proof. We remove $NL1$ from our formula modeling that the TM halts, as $NL1$ with the other axioms forces infinite models. Note that 0 and its s-related successors form a *finite* set of numbers, and we can still encode finite executions of TMs.

However, since our models are finite, we can identify the last element in the s-chain from 0, logically. Let us first demand the following property that says there is one and only one element that has no successor:

$$\exists x.\ (\ (\forall y.\ \neg(s(x,y))) \wedge (\forall z.(\neg(x=z) \Rightarrow (\exists y.\ s(z,y)))))$$

Since the s-chain from 0 has to end in an element that has no successor (since the model is finite), it follows that the maximal element in the chain is the element that has no successor.

Without loss of generality, let us assume that the Turing machine resets its head to the beginning of the tape when it halts (one can modify any TM to do this).

We keep the constraints the same as in the previous reduction, except the formula φ_4. Instead of saying that there is a configuration that halts, we replace it with the following formula φ_5:

$$\forall t.\ (\forall y.(\neg s(t,y))) \Rightarrow \neg H(t,0,1)$$

Note that there is in any computation, the number of cells used by the Turing machine is at most the time it takes on it. So the cell numbers can be bounded by the time take by the Turing machine. The above condition demands that for the last element t in the s-chain, the machine is not in the halting state.

We can show that the TM M does *not* halt iff the formula ψ'_M

$$(NL2 \wedge NL3 \wedge NL4 \wedge \varphi_1 \wedge \varphi_2 \wedge \varphi_3) \Rightarrow \varphi_5$$

is valid over all finite models.

The proof is easy. If the TM halts, we can take a large a finite prefix of natural numbers that at least as long as the number of steps the TM takes to halt, and encode the TM's computation faithfully, and show that the halting state is reached, and hence the formula is not valid.

Conversely, if TM does not halt, then in any finite model that encodes the Turing machine's computation on the s-chain from 0, the final element in the s-chain will not be halting, and hence the formula will be true in the model. Hence the formula is valid.

We hence have:

Theorem 4.3 (Trakhtenbrot's theorem:) *The validity problem for FOL over finite structures is undecidable, and, is in fact, co-r.e.-hard.*

Aside: It is actually possible to do a slightly simpler proof than the above where instead of using a successor relation, we ask that the elements form a total linear order under a relation <, and then recover the successor relation from this ordering relation and constrain 0 *to be the minimal element. Then all elements in the universe are part of the s-chain from* 0 *and we can simply ask all configurations to be non-halting. See Libkin's "Finite Model Theory" book for such a proof. However, we have chosen the above proof since it closely mimics the previous proof that validity over all structures is r.e.-hard. Note that when models are infinite, we can define a linear order, but won't be able to extract a successor relation (it may not even be present as the order can be dense).*

Here is a simple corollary:

Corollary 4.2 *There is no computable function f such that for any sentence φ, if φ has a satisfying finite model, then it has a model of size at most $f(\varphi)$.*

Why? If such a computable function existed, then the satisfiability problem for FO sentences over finite models would be *decidable*, since we can just enumerate all models of size at most $f(\varphi)$, and check whether any of them satisfies the formula (this check is a decidable check). But satisfiability over finite models is undecidable, as validity is undecidable.

Aside: One of the applications of logic is in program verification. When proving programs, you typically want to show a small program snippet has a property, in terms of its input and output, and you can model this often in logic. For programs that manipulate integers, etc., we can use appropriate logics over integers. Program verification, however, is not complete, since programs over integers typically require addition and multiplication, which is not r.e. (we will see this later). Avoiding integers, we could look at programs that manipulate data-structures manipulated by a program that manipulates pointers in heap. But this too turns out to be inherently incomplete as typically data-structures are finite. *Consequently, the validity problem, even after axiomatizing them, is a problem about finite structures that tends to be not r.e.. Testing, such as trying to see if there is some input that takes a program down a path, is easier, as this is a satisfiability problem, that is in r.e.— we can enumerate datastructures, as they are finite, and see whether they drive the program down that path.*

Chapter 5
Quantifier-free theory of equality

In this chapter, we consider the problem of deciding the validity of sentences of the kind

$$\forall x_1, \ldots x_n.\ \varphi$$

where φ is quantifier free, and over an arbitrary signature.

Note that our convention for validity for formulas (with free variables) is that a formula is valid if it is true in every structure and every valuation of variables over the structure. Hence validity of the sentence $\forall x_1, \ldots, x_n.\ \varphi$ is the same as the validity of the formula φ. Since this formula has no quantifiers, this fragment is referred to as the *quantifier-free fragment*. Furthermore, since the functions and relations are not restricted in any way, the only relation that has a fixed interpretation is *equality* (interpretation of = symbol). Hence this theory is called the quantifier-free theory of equality.

We saw in the last chapter that general FOL validity is undecidable. However, the proof of that undecidability crucially used existential quantification (in particular, $\forall\exists$ quantification), and hence that proof does not apply for this fragment. We will show in fact that validity for this fragment is decidable.

Let us consider the dual problem of satisfiability. Given a quantifier-free formula $\varphi(\mathbf{x})$, is there a model and interpretation of $\mathbf{x}$ that satisfies φ? Note that this is the same decision problem as validity, as φ is valid iff $\neg\varphi$ is not satisfiable. Hence a decision procedure for satisfiability gives a decision procedure for validity as well.

5.1 Decidability using Bounded Models

We first make the simple observation that a quantifier-free formula φ is satisfiable iff it is satisfied in a finite model, in fact the finite model needs to be only of size n, where n is the number of *terms* mentioned in the formula.

Let $\varphi(\mathbf{x})$ be a satisfiable quantifier-free formula. Let M be a (potentially infinite) model and s be an interpretation of the variables $\mathbf{x}$, under which $\varphi(\mathbf{x})$ is true, i.e., $M \models \varphi(\mathbf{x})$.

Now let us construct a *finite* model M' from M that also satisfies φ. Let T be the terms mentioned in φ; T is finite, and let's say $|T| = n$. Without loss of generality, assume $n > 0$ (if not, add a conjunction $x = x$ to the formula to introduce a term x). Let the universe in M be U_M. Now, under the interpretation s, each term $t \in T$ evaluates to an element $[t]_s \in U_M$.

Let us define the model M' as follows: the universe U' of M' is $\{[t]_s \mid t \in T\}$ (i.e., the finite subset of elements that terms evaluate to. For every relation symbol R, $R(\overline{u})$ is true in M' iff $R(\mathbf{u}$ is true in M. Functions are a bit more complex to define. Let us fix an arbitrary element e in U'. Define $f^{M'}(\overline{u})$ to be $f^M(\overline{u})$ if $f^M(\overline{u}) \in U'$, and e otherwise.

The above construction basically takes the sub-universe corresponding to the terms mentioned in φ, restricts the relations and functions to this subset, and when the function maps a vector of elements to an element outside this subset, map it to e instead.

Our claim now is that M' with the same interpretation s will also satisfy φ. First, we prove that every term in φ maps to the *same* element in M' as it does in M. And hence any atomic formula $R(\overrightarrow{t})$ as well as any atomic formula $t = t'$ will evaluate the same way under both models. It follows that the formula φ will evaluate to *true*.

The above argument shows that satisfiability for quantifier-free formulas is decidable. We can just enumerate all possible models of size n, where n is the set of terms mentioned in φ, choosing all possible interpretations for functions, relations, constants, and variables, and check if any of them satisfy φ.

If the signature (including arities) are *fixed*, and validity of formulas only over the fixed signature is to be decided, we can even do this in NP. We can just non-deterministically guess the model of size n and the interpretation, and check if the formula is satisfied. Since checking whether a formula holds in the model can be done in polynomial time, this gives an NP algorithm.

It is also easy to see that the problem is NP-hard as well, as it essentially includes Boolean logic. Reduction from SAT: Given a propositional formula α, introduce a new variable x, and replace each proposition occurrence p in φ with the atomic formula $(p = x)$. This formula is satisfiable iff α is satisfiable. Hence satisfiability of quantifier-free formulas is NP-complete.

5.2 An Algorithm for Conjunctive Formulas

The above proof that checking satisfiability is NP-hard is a bit unsatisfactory, as it shows it is hard because of the Boolean logic within in. What is the precise complexity of reasoning with equality itself?

We can ask the above question more precisely by asking what is the complexity of deciding *conjunctive* formulas. Can they be decided in polynomial time?

It turns out that the problem is indeed solvable in polynomial time. We will consider an algorithm in this section that clearly works in polynomial time if the arity of functions/relations are fixed (i.e., bounded by k). However, it turns out that one can implement this algorithm using clever data-structures to get a polynomial time algorithm even when arities are not fixed (see Calculus of Computation, Chapter 9, Section 9.3, for example).

To simplify the algorithm, let us first get rid of predicates (other than equality) from the formula. This can be done easily. Let us fix a constant $\mathbb{T}$. We can model a predicate $p \subseteq U^n$ as a *function* $f_p : U^n \rightarrow U \cup \{T\}$, with the understanding that $p(\overline{u})$ is true iff $f_p(\overline{u}) = \mathbb{T}$. Hence, given a formula φ with functions and relations, we can replace each occurrence of $p(\bar{t})$ with $f_p(\bar{t}) = \mathbb{T}$, to get a formula φ' such that φ is satisfiable iff φ' is satisfiable. (We leave this as an exercise.)

Let φ be a conjunctive formula that is quantifier-free, that uses function symbols and =, but no relation symbols. Then $\varphi \equiv \alpha_1 \wedge \alpha_2 \wedge \ldots \alpha_n$, where each α_i is a literal of the form $t = t'$ or $\neg(t = t')$.

Since the formula is conjunctive, let us look upon the formula as a set of conjuncts: $\{\alpha_1, \ldots, \alpha_n\}$. In fact, let us divide these formula into two sets $E \cup D$, where E consists of equalities of the form $t = t'$ and D consists of disequalities of the form $\neg(t = t')$. Let us look upon the elements of E and D as pairs of terms of the form (t, t').

Our key idea is now to build a model, if the formula is satisfiable, just using the terms T mentioned in the formula φ, which is finite and linear in $|\varphi|$. Furthermore, our idea is to find the *smallest* set of equality constraints that are imposed by the set of equalities in E. It turns out that such a smallest set always exists, and is called the *congruence closure of* E, denoted $CC(E)$, and is easy to compute in polynomial time. We then check whether any of the disequalities are violated in $CC(E)$. Then there is a violation iff the formula is not satisfiable.

The *congruence closure of* E, $CC(E)$ is the smallest set such that:

Includes E: For every $(t, t') \in E$, $(t, t') \in CC(E)$
Reflexive closure: For every $t \in T$, $(t, t) \in CC(E)$
Symmetric closure: For every $t, t' \in T$, if $(t, t'), (t', t) \in CC(E)$.
Transitive closure For every $t, t', t'' \in T$, if $(t, t') \in CC(E)$ and $(t', t'') \in CC(E)$, then $(t, t'') \in CC(E)$.
Congruence closure For every function symbol R of arity n, for every $f(t_1, \ldots, t_n), f(t'_1, \ldots, t'_n) \in T$, if $(t_1, t'_1), (t_2, t'_2), \ldots (t_n, t'_n) \in CC(E)$, then $(f(t_1, \ldots, t_n), f(t'_1, \ldots, t'_n)) \in CC(E)$

Intuitively, $CC(E)$ is the reflexive, symmetric and transitive closure of E, and also congruence-closed, in the sense that if two tuples of terms are deemed equal by it, then the function expressions applied on those terms are also deemed equal.

It turns out that $CC(E)$ exists (i.e., a smallest set of such equalities exists). Here is a simple proof. Define

$$\begin{aligned}
CC_0 &= E \\
CC_{i+1} &= CC_i \\
&\qquad \cup \ \{(t,t) \mid t \in T\} \\
&\qquad \cup \ \{(t,t') \mid (t',t) \in CC_i\} \\
&\qquad \cup \ \{(t,t') \mid \exists t'' \in T, (t,t''), (t'',t') \in CC_i\} \\
&\qquad \cup \ \{(f(t_1,\ldots,t_n), f(t'_1,\ldots,t'_n)) \mid (t_1,t'_1),\ldots,(t_n,t'_n) \in CC_i, f(t_1,\ldots,t_n), f(t'_1,\ldots,t'_n) \in T\}
\end{aligned}$$

Now, let $CC = \cup_{i \in \mathbb{N}} CC_i$. Then it is easy to prove that CC has the required properties and is the smallest set that has these properties (readers should prove this for themselves). Since T is finite, the above procedure in fact terminates, i.e., there will be an i, such that $CC_i = CC_{i+1}$, in which case we can stop, as the future sets will all be the same. It's easy to see that this is in fact a polynomial time algorithm, since CC_i monotonically increase and can have at most $|T|^2$ pairs. We will see later in this section a concrete algorithm that does this computation a bit better.

Now let M be any model that satisfies the equalities in E. Then it is clear that M will satisfy the equalities in $CC(E)$ as well, since what we demand of $CC(E)$ are properties satisfied by equality. Consequently, the equalities in $CC(E)$ are logically implied by the equalities in E.

Let us now prove:

Lemma 5.1 *There exists a model satisfying the equalities in E and the disequalities in D exist iff* $CC(E) \cap D = \emptyset$

Proof In the forward direction, assume M is a model satisfying the equalities in E and the disequalities in D. Then, as we argued above, the equalities in $CC(E)$ must be satisfied in M as well, as they are logically implied by the equalities in E. It follows that since every disequality in D is satisfied by M, $CC(E)$ and D cannot have a common pair of terms.

In the other direction, assume $CC(E)$ and D are disjoint. Let us define the equivalence relation $\sim$ over T defined by $CC(E)$ as $t \sim t'$ iff $(t,t') \in CC(E)$. It is easy to see that $\sim$ is indeed an equivalence relation.

Let us construct a model M, where the universe U of M is $T/\!\sim$, i.e., the universe is the set of equivalence classes of $\sim$. Interpret each constant symbol c occurring in φ as the equivalence class $[\![c]\!]$, and each variable x occurring in φ as the equivalence class $[\![x]\!]$. The interpretation for a function symbol f on a vector of elements $e_1, \ldots e_n$ is defined as follows (for the definition below, fix a particular element e^* arbitrarily):

- If there are some terms $t_1 \in e_1, \ldots t_n \in e_n$ such that $f(t_1,\ldots,t_n)$ is a term in T, then map $f(e_1,\ldots,e_n)$ to $[\![f(t_1,\ldots t_n)]\!]$.
- Else, map $f(e_1,\ldots,e_n)$ to e^*.

Note that the above model construction is well defined only because $CC(E)$ satisfies the congruence-closure condition. For example, if $t_1 \sim t_2$, then we would need $f(t_1) \sim f(t_2)$ in order for the definition of f to be well-defined.

It is now straightforward to argue, by induction on structure of terms, that for every term t, the term evaluates to the element $[t]$ in this model. Consequently all the equalities E are satisfied. Also, every disequality $(t, t') \in D$, we are guaranteed $[t] \neq [t']$, since $CC(E) \cap D = \emptyset$. Hence the formula holds in the model, and is hence satisfiable. □

The above shows that in order to check whether a quantifier-free conjunctive formula φ is satisfiable, we just need to compute $CC(E)$ and check if $CC(E) \cap D = \emptyset$, where E and D are the equalities and disequalities occurring in φ.

5.2.1 Computing $CC(E)$

One simple method for computing the congruence closure, especially on paper manually, is to compute successive equivalence relation using its *equivalence* classes.

An equivalence relation $\sim$ over T can be represented as a set of sets that form a partition of T, i.e., as $\{E_1, \ldots E_k\}$ where each $E_i \subseteq T$, the sets are all disjoint, and their union is E_i.

Given such a representation of $\sim$, let us define a *Merge* operation on them: $Merge(\sim, E_i, E_j)$, where E_i, E_j are two equivalence classes of $\sim$ simply merges the two equivalence classes into one and results in a new equivalence relation. More precisely, if $\sim$ is $\{E_1, \ldots, E_k\}$, then $Merge(\sim, E_i, E_j)$ is $\sim'$ whose representation is $\{E_r \mid r \neq i, r \neq j\} \cup \{E_i \cup E_j\}$.

The algorithm for computing congruence closure is then as follows. :

- Initialize $\sim$ to $\{\{t\} \mid t \in T\}$. In other words, we start with each term in its own equivalence class.
- For every $(t, t') \in E$, merge $[t]$ and $[t]'$.
- Do the following till $\sim$ stabilizes:
  - If there are any terms $t_1, \ldots, t_n, t'_1, \ldots, t'_n \in T$ such that both $f(t_1, \ldots, t_n)$ and $f(t'_1, \ldots t'_n)$ are in T, and if $[f(t_1, \ldots, t_n)]$ is not equal to $f(t'_1, \ldots, t'_n)$, then merge them.
- Check whether there is any disequality $(t, t') \in D$ such that $[t] = [t']$; if there is, report formula is unsatisfiable; otherwise, report formula is satisfiable.

Let us illustrate through an example:

Example 5.1 This example is taken from the book Calculus of Computation.

We want to know whether the following formula is satisfiable:

$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$

Equalities E are $\{(f(a, b), a)\}$.
Disequalities D are $\{(f(f(a, b), b), a)\}$
The set of terms T is $\{a, b, f(a, b), f(f(a, b), b)$.

We start with the equivalence relation:

$$\{\ \{a\}, \{b\}, \{f(a, b)\}, \{f(f(a, b), b)\}\ \}$$

Since $(f(a,b),a)$ are in E, we merge their equivalence classes to get:

$$\{\ \{a, f(a,b)\}, \{b\}, \{f(f(a,b),b)\}\ \}$$

Since a and $f(a,b)$ are in the same equivalence class, and since b and b are in the same equivalence class, we merge $f(a,b)$ and $f(f(a,b),b)$ to get:

$$\{\ \{a, f(a,b), f(f(a,b),b)\}, \{b\}\}\ \}$$

It is easy to verify that the equivalence classes have stabilized.

We can now check whether the disequalities are all satisfied. But since $f(f(a,b),b)$ and a are in the same equivalence class, we report that the formula is unsatisfiable.

Let us now illustrate an example where the formula is satisfiable, and also illustrate the model construction involved in the Lemma above.

Example 5.2 We want to know whether the following formula is satisfiable:

$$f(a) = b \wedge f(b) = a \wedge f(f(a)) = c \wedge \neg(f(a) = a)$$

The equalities are: $E = \{(f(a),b), (f(b),a), (f(f(a)),c)\}$.
The disequalities are: $D = \{(f(a),a)\}$
The terms are $T = \{a, b, c, f(a), f(b), f(f(a))$.

We start with the equivalence relation:

$$\{\ \{a\}, \{b\}, \{c\}, \{f(a)\}, \{f(b)), \{f(f(a))\}\ \}$$

Since $(f(a),b)$ is in E, we merge their equivalence classes to get:

$$\{\ \{a\}, \{b, f(a)\}, \{c\}, \{f(b)\}, \{f(f(a))\}\ \}$$

Since $(f(b),a)$ is in E, we merge their equivalence classes to get:

$$\{\ \{a, f(b)\}, \{b, f(a)\}, \{c\}, \{f(f(a))\}\ \}$$

Since $(f(f(a)),c)$ is in E, we merge their equivalence classes to get:

$$\{\ \{a, f(b)\}, \{b, f(a)\}, \{c, f(f(a))\}\ \}$$

Now, since b and $f(a)$ are in the same equivalence class, we must merge the equivalence classes of $f(b)$ and $f(f(a))$. We get:

$$\{\ \{a, f(b), c, f(f(a))\}, \{b, f(a)\}\ \}$$

The equivalence class has now stabilized. We note that there is only one disequality, $(f(a),a)$, and $f(a)$ and a are in different equivalence classes. So we report the formula to be satisfiable.

Let us examine now how the proof of the Lemma above constructs a model. We have two elements in our model, e_1 standing for the class $\{a, f(b), c, f(f(a))$ and e_2 for the class $\{b, f(a)\}$.

Since a is in e_1 and $f(a)$ is in e_2, we interpret that $f(e_1) = e_2$. Also, since $f(a)$ is in e_2 and $f(f(a))$ is in e_1, we interpret $f(e_2) = e_1$. The constants a and c are interpreted as e_1 (since they belong to e_1) and the constant b is interpreted as e_2 (as b belongs to e_2. This model satisfies the formula.

There are even faster ways to do congruence closure. Notice that the key operations above have to manipulate disjoint sets and support union (merge) and find (which equivalence class does an element belong to?). This turns out to be a well-studied data structure called *disjoint-set datastructure* (or *union-find datastructure* for which efficient algorithms are known. Note that the number of equivalence classes is n, where n is the number of terms in the formula, which is of course linear in the size of the formula. Now if the signature is finite and fixed, or if the signature had functions of fixed arity (the former implies the latter), then it is easy to see that the above algorithm can be implemented in polynomial time. If the maximum arity is k, then there are only n^k possible considerations of terms to consider for identifying candidates of equivalence classes to merge. If the signature consists of functions of arbitrary arity, it turns out that one can still effect a polynomial time algorithm, but we skip this here.

SMT solvers also implement fast algorithms for congruence closure. In particular, given a quantifier-free formula (not necessarily conjunctive), they look upon the formula as a Boolean formula over propositions (each proposition being an atomic formula), and call a SAT solver to find a satisfying valuation (if the SAT solver finds it unsatisfiable, then clearly the original formula is also unsatisfiable). The satisfying valuation can now be interpreted as a *conjunctive* set of equality and disequality constraints, which can then be checked for satisfiability using congruence-closure algorithms. If this conjunctive formula is unsatisfiable, it would return a *core* set of atomic formulas that already make it unsatisfiable, and the SAT engine will add that as a clause, and continue its search for valuations.

5.3 Axioms for The Theory of Equality

In this book/course, we will treat the equality symbol (=) as an interpreted relation throughout— it is interpreted as equality of elements in the universe. However, in this section, we are briefly going to suspend that in order to understand what properties equality really satisfies.

Let us fix a FO signature, and for clarity, let us not have = as a symbol, but instead have the symbol $\doteq$. If $\doteq$ was uninterpreted, what properties would we like it to satisfy?

Here are some properties (which we will call the *congruence axioms*) that equality clearly satisfies (we continue to write relations as $t \doteq t'$)), instead of $\doteq(t, t')$:

Reflexivity" $\forall x.\ x \doteq x$
Symmetry: $\forall x.\ (x \doteq y \Rightarrow y \doteq x)$
Transitivity: $\forall x, y, z.\ (x \doteq y \wedge y \doteq z) \Rightarrow x \doteq z$
Congruence wrt relations: For any relation R of arity n,

$$\forall x_1, \ldots x_n, y_1, \ldots, y_n.\ \left(\left(\bigwedge_{i \in [1,n]} x_i \doteq y_i\right) \Rightarrow R(x_1, \ldots, x_n) \Leftrightarrow R(y_1, \ldots, y_n)\right)$$

Congruence wrt functions: For any function f of arity n,

$$\forall x_1, \ldots x_n, y_1, \ldots, y_n.\ \left(\left(\bigwedge_{i \in [1,n]} x_i \doteq y_i\right) \Rightarrow f(x_1, \ldots, x_n) \doteq f(y_1, \ldots, y_n)\right)$$

First, notice that the above doesn't ensure that $\doteq$ will be interpreted as *true* equality on the model. For example, if there were two elements e_1 and e_2 such that all functions and relations behaved identically on them and no constant was interpreted as either of them, then we could relate them with $\doteq$ and satisfy all the properties above. In fact, FO with $\doteq$ (and without $=$) will not be able to distinguish this model from one where we removed e_2 and just had e_1. The above properties in fact only insist that $\doteq$ is an equivalence relation that is also a congruence with respect to the relations and functions.

However, it turns out that the above properties are sufficient to capture equality as far as satisfiability/validity of logical formulae are concerned. It doesn't matter that the properties above capture only congruence and not true equality.

Let us formalize this. Let M be a model with universe U an interpretation of the relation $\doteq$ that satisfies the congruence axioms given above. Then let $M/\doteq$ be the model where we take as the universe the equivalence classes $U/\doteq$, and interpret relations and functions as follows:

- For any constant c,
$$c^{M/\doteq} = [c^M]$$
- For any n-ary relation R,
$$R^{M/\doteq}([e_1], [e_2], \ldots [e_n]) \textit{ holds iff } R^M(e1, \ldots, e_n) \textit{ holds}$$
- For any n-ary function f,
$$f^{M/\doteq}([e_1], [e_2], \ldots [e_n]) = [f^M(e_1, \ldots, e_n)]$$

The above says that constants are interpreted in $M/\doteq$ as the equivalence class of their interpretation in M, and relations and functions are interpreted in $M/\doteq$ depending on how the relations and functions are interpreted on the elements in their equivalence classes. The reason the above is well-defined is because $\doteq$ satisfies the congruence axioms.

We can now show the following:

Lemma 5.2 *Fix a signature S without $=$ and $\doteq$. Let φ be a formula over $S \cup \{=\}$. Let φ' be φ, where $=$ is replaced with $\doteq$, and hence is over the signature $S \cup \{\doteq\}$.*

- *If φ holds in a model M, where $=$ is interpreted as equality in the model, then φ' holds in M' where $\doteq$ is interpreted as equality, and the interpretation of $\doteq$ does satisfy the congruence axioms.*
- *If φ' holds in a model M, where $\doteq$ satisfies the congruence axioms, then φ holds in $M/\doteq$ with $=$ interpreted as equality in the model.*

We leave the above as an exercise for the reader.

A consequence of the above lemma is that a formula with $=$ is satisfiable (or valid) iff the formula, with $=$ replaced by $\doteq$ is satisfiable (or valid) over the class of models that satisfy the congruence axioms.

Hence the above congruence axioms *define* equality as far as logic goes. Logically, there is no real difference between true equality and a congruence.

Chapter 6
Completeness Theorem: FO Validity is r.e.

Gödel proved in 1929 his first famous theorem that there is a formal proof system that can prove every valid formula in FOL. As the formal proof system one can choose a variety of proof systems (Gödel showed it for one proposed by Hilbert and Ackermann). A proof system is a formal set of rules of writing down a finite sequence (called a proof) that establishes the validity of a formula/sentence. In fact, a stronger statement is proved (let's call this the strong completeness theorem): there is a formal proof system such that for any set of axioms A, the formal system (incorporating axioms A) can prove any formula/sentence that is semantically entailed by A. In other words, the system can prove any sentence φ where φ holds in *all* models that satisfy the axioms A.

The above result is remarkable. It basically shows that any theorem that can be stated in FOL has a proof. Not only that, for any class of axiomatizable structures, the class of valid FO-formulatable theorems over such structures always has a proof. For instance, take the class of groups— they can be axiomatized using a few axioms, as we saw in Chapter 1. Consequently, every first-order formulatable theorem over groups has a proof.

Given a set of formulae/sentences A, the validity problem for the theory of A is to determine whether, given a formula/sentence φ, whether every model and every interpretation under which A holds also satisfies φ.

Connection to computability

In this book/course, we won't be studying proof systems, and hence won't prove Gödel's completeness theorem. However, we will prove essentially Gödel's completeness theorem, but where we replace proofs with *computation*.

Consider a set of axioms A which is a decidable set (i.e., it is either empty or finite or infinite where a TM can check whether a given sentence is an axiom or not). Then Gödel's completeness theorem says that every logically entailed theorem has a proof. Proofs are generally *finite* objects— they are typically finite sequences over some signature, that closely follow some set of allowed rules, incorporating the axioms when needed, in order to prove a theorem. Whatever the proof system is, it is always true that *checking* whether such a sequence encodes a correct proof is a decidable problem. Consequently, it is easy to see that validity with respect to the axioms is a problem solvable in r.e.. This is because a Turing machine can enumerate all possible finite proofs, systematically, checking if any of them prove a given theorem, and halt if it does. So Gödel's theorem can be seen as saying that the problem of checking validity wrt any recursive set of axioms is recursively enumerable.

Our goal in this chapter is hence to prove this version of completeness. For every formula/sentence, when the TM finds that the sentence is a theorem in the theory of the given axioms, the computation itself can be viewed as a proof of the theorem.

Outline of Proof

The procedure we are going to outline is not entirely the usual one found in standard textbooks, and has a more computational flavor. As we will see, it can also be automated to some extent (we can build an r.e. procedure using calls to an SMT solver that decides the quantifier-free theory of equality).

The rough outline is as follows. We fix a countable signature. We are given a countable decidable set of axioms A and we consider the problem of proving validity of a FOL formula ψ.

1. Our procedure will work through *refutation*— we will show that $\psi' = \neg\psi$ is not satisfiable in any model satisfying the axioms. In other words, we want to show there is no model satisfying $A \cup \{\psi'\}$.
2. We first show that formulas can be translated to equivalent formulas in *prenex normal form*. Then we show that we can convert the negated formula to an *equisatisfiable* formula ψ'' over an expanded signature which has only universal quantification, and is of the form

$$\psi'' : \forall \overline{x}.\ \varphi$$

   This process is called *Skolemization*.
3. We then show *Herbrand's Theorem* for such sets of universally quantified formulas, which roughly says that if the axioms and the formula is satisfiable, then they satisfiable in a universe that is composed of only ground terms over the signature modulo a congruence.
4. The above result will show that the universally quantified formula will be *unsatisfiable* iff replacing variables with all possible terms, which gives an infinite set of formulas, is an unsatisfiable set.
5. We will then use compactness of propositional logic to argue why this instantiated infinite set is unsatisfiable iff there is a finite subset of it that is unsatisfiable.
6. The above gives our r.e. procedure: negate the formula, Skolemize the axioms and formula, and instantiate systematically the formulas by a growing set of terms and

check whether the the resulting set is unsatisfiable. Any instantiation procedure that dovetails between the axioms and term instantiation so that all axioms are instantiated for all terms eventually will do. Each level of instantiation gives a set of quantifier-free formulae in the theory of equality, which is decidable. The algorithm will halt only if it finds that there is some level where the instantiated set is unsatisfiable.

We first show Step 2: Skolemization. Then we prove Herbrand's theorem. We then will use compactness to argue unsatisfiability can be proved using only a finite set of terms. And finally give the r.e. procedure.

6.1 Prenex Normal Form

We assume that the formulae/sentences we are considering for validity/satisfiability have first been convereted to prenex normal form, i.e., to the form:

$$Q_1 x_1.\, Q_2 x_2 \ \ldots Q_n x_n.\varphi$$

where φ is quantifier free, and furthermore, where no variable repeats (for every $i \neq j, x_i \neq x_j$).

We refer the reader to a standard text that shows that any formula in FOL can be converted to an equivalent formula in prenex form.

6.2 Skolemization / Herbrandization

Recall that for validity, pure universal quantification was easy to handle (we showed decidability in the last chapter). Hence, for satisfiability, pure existential quantification is easy to handle.

We can in fact eliminate all existential quantification in a satisfiability problem easily.

Consider a formula of the form

$$\psi : \forall x_1, \ldots x_n.\exists y.\varphi(x_1, \ldots, x_n, y)$$

where φ is an arbitrary formula (can have quantifiers). We will show that there is an equisatisfiable formula (over an expanded signature) where we essentially remove the quantified variable x.

The formula roughly says:

For every valuation of $x_1, \ldots, x_n$, there exists a value for y such that φ holds.

Assume there is some model that satisfies the above property. Then for every sequence of values of $x_1, \ldots, x_n$, since there is an element y in the universe such that φ holds, we can fix a particular choice of this element y using a new *function* f. This function in the model takes a tuple of values $(v_1, \dots, v_n) \in \mathcal{U}^n$ (standing for a valuation of $x_1, \dots, x_n$, respectively) and maps it to an element in $\mathcal{U}$. Now, instead of saying that there is a value y that satisfies φ, we can instead say that choosing y to be $f(x_1, \dots, x_n)$ satisfies φ.

More precisely, we can write instead the formula:

$$\psi' : \forall x_1, \dots x_n.\ \varphi(x_1, \dots, x_n, f(x_1, \dots, x_n) \ / \ y)$$

In other words, we remove the existential quantification on y, and instead replace y in φ with $f(x_1, \dots x_n)$. Here, f is a *new* function symbol introduced specifically for this quantification of y.[1].

It should be clear that the original formula ψ is satisfiable over a signature Σ iff the above formula ψ' is satisfiable over the signature $\Sigma \cup \{f\}$, where f is a function symbol not occurring in Σ. In the forward direction, if ψ is satisfiable in a model M, we construct a model M' over the expanded signature that extends M by interpreting f on an n-tuple of values to some value y that makes φ true when $x_1, \dots, x_n$ are evaluated as the n-tuple. This extended model M' will satisfy ψ'. In the reverse direction, if there is a model M' for ψ', we can show that the model M which is the same as M' except with the interpretation of f erased, satisfies ψ: for every valuation of $x_1, \dots, x_n$, if we choose choose y to be $f(x_1, \dots, x_n)$, then we are guaranteed to satisfy φ.

When a formula has no universal quantification preceding an existential quantifier, the above works too, except that now the function takes *no arguments*, i.e., it is a 0-ary function. A function that takes no arguments and returns an element is essentially a constant. So we can replace such an existentially quantified variable with a new constant symbol.

More precisely, we can show that for any formula $\exists x.\varphi$ over a signature Σ, the formula $\varphi[c/x]$, where c is a new constant symbol that is not in Σ, is equisatisfiable.

The following lemma captures the above:

Lemma 6.1 *For any formula $\psi : \forall x_1, \dots x_n.\exists y.\varphi(\overline{z}, x_1, \dots, x_n, y)$ over a signature Σ, let f be a function symbol not in Σ, and let*

$$\psi' : \forall x_1, \dots x_n.\ \varphi(\overline{z}, x_1, \dots, x_n, f(x_1, \dots, x_n) \ / \ y)$$

over the signature $\Sigma \cup \{f\}$, where the arity of f is n. Then ψ and ψ' are equisatisfiable.

Also, for any formula $\psi : \exists y.\varphi(\overline{z}, y)$ over a signature Σ, let c be a constant symbol not in Σ, and let

$$\psi' : \varphi(\overline{z}, c \ / \ y)$$

over the signature $\Sigma \cup \{c\}$. Then ψ and ψ' are equisatisfiable.

Example 6.1 For example, consider the formula

[1] Some readers may wonder if we are using the axiom of choice here; we are.

$$\forall x.\exists y.R(x,y)$$

which says that every element x is R-related to *some element* The above is equisatisfiable to the formula

$$\forall x.R(x,f(x))$$

Intuitively, the function f chooses one of the (potentially several) elements x is R-related to. Such a function exists iff every x is indeed R-related to some element.

We can now apply the above procedure of eliminating existential quantifiers repeatedly to a formula in prenex rectified normal form in order construct a purely universally quantified formula that is equisatisfiable.

Let us call formulas that are purely universally quantified *universal formulas*.

Herbrandization:

The above also shows that we can take any formula ψ and convert it into an equi-valid formula of the form $\exists x_1. \ldots, \exists x_n \varphi$ over an expanded signature. We can simply take $\neg psi$, Skolemize it to derive an equi-satisfiable formula over an expanded signature, and then take its negation, to get a formula with purely existential quantification that is equi-valid to ψ. This process of getting equi-valid formulas with existential quantification only is called Herbrandization.

6.3 Herbrand's theorem

One of the the first hurdles for solving satisfiability or proving unsatisfiability is to figure out what the *universe* for a formula might be. Clearly, we need elements to represent constants as they are terms. And we need elements to represent terms formed by applying functions (any number of times) to terms. But do we need more? Can a formula/sentence talk about elements that are are not *accessible* by using functions that involve constants?

Let us define accessible elements more formally. A *ground term* is a term without variables (it it built only using functions and constants). Let M be a model. An element e in the universe of M is said to be *accessible* if there is a ground term t such that t evaluates to the element e in the model M. A model is said to be *fully accessible* if every element of it is accessible.

We can now ask whether every sentence that is satisfiable has a fully accessible model. It turns out this is not true. For example, consider a signature that has a constant 0 and a function s and the formulae that force a number line from 0:

$$\varphi_0 : \ \forall x.\ (\neg s(x) = 0) \land \forall x, y.\ (s(x) = s(y) \Rightarrow x = y)$$

For this formula, it is indeed true that it is satisfied in a fully accessible model, for example a model that contains elements that serve as interpretations for $0, s(0), s(s(0)), \ldots$ only.

However, consider adding a conjunct:

$$\varphi_1 : \ \varphi_0 \wedge \forall x.\ \exists y.(\ f(y) = x \wedge s(y) = y)$$

This formula means that there must be elements whose f images are $0, s(0), s(s(0))$, etc., and hence these elements must all be *distinct* as well. These have to be different from the 0-chain and must be distinct from each other as well (as their f-images are different). Note that there are no *ground terms* that access these (infinitely many) elements. For example, one model that satisfies the above constraints is:

$$U = \mathbb{N} \cup \{i' \mid i \in \mathbb{N}\}$$

$$s(i) = i + 1, \textit{ for every } i \in \mathbb{N}$$

$$s(i') = i', \textit{ for every } i \in \mathbb{N}$$

$$f(i') = i, \textit{ for every } i \in \mathbb{N}$$

$$f(i) = i, \textit{ for every } i \in \mathbb{N}$$

Note that there are no ground terms that "evaluate" to the elements i', where $i \in \mathbb{N}$.

It turns out however that *universal formulae do indeed have the property that satisfiable sentences always have fully accessible models*. This is called *Herbrand's theorem* which we will prove below.

In fact, in the above example, the formula φ_0 is a universal sentence and has a fully accessible model. The sentence φ_1 does not have a fully accessible model, and notice that it uses an existential quantifier. We can, as argued in the last section, Skolemize formulas to have an equisatisfiable formula that has only universal quantification. Skolemizing φ_1 gives:

$$\varphi_1' : \ \varphi_0 \wedge \forall x.\ (\ f(g(x)) = x \wedge s(g(x)) = g(x)\ )$$

Notice that the Skolemization introduces a new function g for the existentially quantified variable y removed. And notice now there is a satisfying fully accessible model. In the model above sketched, we can make $g(i) = i'$ to satisfy the formulas (g for other elements can be defined arbitrarily).

Note that having accessible models is very pleasing. The universe can be thought of as consisting *only* of ground terms in the logic! In fact, we can *name* our elements using the terms in the logic (more precisely, equivalence classes of terms will be the elements in our universe). Also, notice that if ground terms $t_1, \ldots, t_n$ access the elements $e_1, \ldots, e_n$, respectively, in a model M, then clearly $f(t_1, \ldots, t_n)$ accesses the element $f^M(e_1, \ldots, e_n)$. Consequently, in the models we build, the interpretation of functions is *fixed*— the function f will map the ground terms $t_1, \ldots, t_n$ (which are in the universe as the universe consists only of ground terms) to the ground term $f(t_1, \ldots, t_n)$. So, really, the universe, and the interpretation of constants and functions will be *fixed*. The only things to figure out are the interpretation of relations, including equality which will cause the universe to be equivalence classes over ground terms.

We now prove that this is always the case— universal sentences that are satisfiable have fully accessible models. More precisely, we will define Herbrand models where elements *are* equivalence classes of ground terms (with fixed interpretations of functions), and show that satisfiable universal sentences (also called sentences in Skolem form) have Herbrand models.

Universal Formulas and Closed Submodels

The key property that universal sentences satisfy is that they are satisfied in any *submodel* of a satisfying model, as long as the submodel is closed with respect to function applications. Let φ be a universal sentence and M, with universe U, be a model satisfying it. Let $U' \subseteq U$ that satisfies the following properties:

- For every constant c, $c^M \in U'$
- For every function symbol of arity n, if $e_1, \ldots, e_n \in U'$, then $f^M(e_1, \ldots, e_n) \in U'$.

Then the submodel $M' = (U', I')$ define by taking U' as the universe and interpreting all constants, functions, and relation symbols on U' exactly as in M, but restricted to U', is called a *closed submodel*. More precisely, we define the interpretation of the closed submodel with universe U' to be:

- $c^{M'} = c^M$, for every constant symbol c
- For every relation symbol R of arity n, and for every $e_1, \ldots, e_n \in U'$, $R^{M'}(e_1, \ldots, e_n)$ *iff* $R^M(e_1, \ldots, e_n)$
- For every function symbol f of arity n, and for every $e_1, \ldots, e_n \in U'$, $f^{M'}(e_1, \ldots, e_n) = f^M(e_1, \ldots, e_n)$

Note that the properties that U' needs to satisfy is crucial to build the submodel— we cannot build a submodel using any sub-universe of elements (the values that f takes tuples of elements in the sub-universe to must be in the sub-universe as well).

If M' is a closed submodel of M, it turns out that M' will satisfy all the universal sentences that M satisfies (the converse does not hold, of course). Note that a sentence that has an existential quantification, say of the form $\exists x.R(x)$, may hold in M but may not hold in M' (as the elements witnessing the property may be not in the sub-universe, for example). But satisfiability is preserved for submodels on universal formulas. The proof is rather simple:

Lemma 6.2 (Closed submodel property) *Let M be a model and let M' be a closed submodel of M. Let φ be a universal sentence and let $M \models \varphi$. Then $M' \models \varphi'$. Furthermore, every term evaluates to the same element in M' as it does in M.*

Proof Fix a model M, a closed submodel M', and a universal sentence φ : $\forall x_1, \ldots, x_n.\varphi'$ where φ' is quantifier free, where $M \models \varphi$. Let $e_1, \ldots, e_n$ be the interpretation of the variables $x_1, \ldots, x_n$ in the universe of the submodel M'. Then these belong to the universe in M, and since the universe of M' is closed under function applications, and since M' inherits the interpretations of constants and functions from M, it follows that every term t involving constants and these variables evaluate to the same element in M' as they do in M. Since M' also inherits the relations from M (including equality), it follows that all atomic formulas involving these variables evaluate to the same Boolean value in M and M'. Since φ' is quantifier-free, it too will evaluate to the same value in M as in M'. Since M' satisfies φ, for this interpretation of variables, φ' will also evaluate to *true*. Hence we have shown that for all possible interpretations of the variables in the universe of the submodel, φ' evaluates to *true*, which means that $M' \models \varphi$. □

Note in the above that we don't make the claim for *universal formulas* but just for *universal sentences*. The reader should make sure they understand why the lemma does not hold for universal formulas.

Herbrand Models and Herbrand's Theorem

Let us now define Herbrand models.

Definition 6.1 Fix a FO signature Σ. Let GT be the set of all ground terms over Σ. A *functional congruence over ground terms* $\sim$ is an equivalence relation over ground terms such that for every $t_1, \ldots, t_n, t'_1, \ldots, t'_n$, where for each $1 \leq i \leq n$, $t_i \sim t'_i$, it is the case that $f(t_1, \ldots, t_n) \sim f(t'_1, \ldots, t'_n)$. For such a congruence $\sim$, we denote the equivalence class containing t with the notation $[\![t]\!]_\sim$.

In the notation for equivalence classes, we sometimes write $[\![t]\!]$, if $\sim$ is clear from context.

Definition 6.2 (Herbrand model with equality) Fix a FO signature Σ with at least one constant symbol (hence the set of ground terms over Σ is non-empty). A Herbrand model (with equality) is one where:

- The universe of the model is $U = \{[\![t]\!] \mid t \in GT\}$ consists of the set of equivalence classes of ground terms of Σ with respect to some functional congruence over ground terms $\sim$.
- Any constant c is interpreted as $[\![c]\!]$.
- Any function symbol f of arity n is interpreted so that for every $t_1, \ldots, t_n \in GT$, $f^M([\![t_1]\!], \ldots, [\![t_n]\!]) = [\![f(t_1, \ldots, t_n)]\!]$.

The first condition says that the universe of a Herbrand model consists of just equivalence classes of terms with respect to a functional congruence $\sim$. The second says that the interpretation of functions is given by the names of the elements themselves— a function f will take the equivalence classes of n terms $t_1, \ldots, t_n$ to the equivalence class of the term $f(t_1, \ldots, t_n)$. This definition of f^M is well-defined since $\sim$ is a functional congruence over terms.[2]

Let us make some simple observations. First, in a Herbrand model, because of the way constants and functions are interpreted, it is easy to show, by induction, that every ground term t evaluates to the equivalence class containing it, i.e., $[\![t]\!]$. It hence follows that in a Herbrand model is fully accessible— every element $[\![t]\!]$ is accessible using the term t.

In fact the converse is also true: every fully accessible model is a Herbrand model, which will be evident in the proof of Herbrand's theorem below.

Let us now prove Herbrand's theorem. [3] Herbrand's theorem states that if a universal sentence has a model, it has a Herbrand model.

The intuition of the proof is quite simple. Let M be a model satisfying a universal sentence φ. Then we can simply take the sub-universe that corresponds to all accessible elements (elements accessible using terms). Clearly, this subset is closed under function applications. And hence it defines a closed submodel that satisfies φ as well. This closed submodel, having accessible elements only, is isomorphic to a Herbrand model— we can relabel every element e using the equivalence class of *all ground terms that evaluate to the element e*, in order to make it a Herbrand model.

Theorem 6.1 (Herbrand's theorem with equality) *Let φ be a universal sentence. Then φ is satisfiable iff it is satisfiable in a Herbrand model.*

Proof We prove the forward direction (the reverse direction is trivial as if φ has a Herbrand model, then it is clearly satisfiable).

Let φ be satisfiable. Let M be a model for φ, with universe U.

Let $U' = \{t^M \mid t \text{ is a ground term }\}$. Then, clearly, U' satisfies the properties for defining a closed submodel of M— it includes the interpretations of all constant symbols, and for any function symbol of arity n and any n-tuple of elements, say $t_1^M, \ldots, t_n^M$, it clearly contains $f^M(t_1^M, \ldots, t_n^M)$, as that is precisely $f(t_1, \ldots t_n)^M$.

Now let M' be the closed submodel of M defined by U'. By the previous lemma, $M' \models \varphi$.

We now prove M', with its elements renamed, is in fact a Herbrand model. Define a congruence on ground terms as follows: $t \sim t'$ iff $t^M = t'^M$. Verify that this is indeed a congruence on ground terms (proof: verify it is an equivalence relation, and note that if $t_1, \ldots, t_n, t'_1, \ldots, t'_n$ are such that each for each i, $t_i \sim t'_i$, then

[2] If you were a student in elementary school and you knew Herbrand models, and your math teacher asked you what 2 + 3 is, you would say it's "2 + 3"! The value of the function + applied on the terms $\langle 2, 3\rangle$ is simply the term/element $+(2, 3)$. You may not pass your elementary school exams though!

[3] Herbrand's theorem is generally proved in a signature without equality. Then one can show that purely universally quantified formulas have a model where the elements are terms, not equivalence classes of terms. Since we want to treat equality as an interpreted relation that is always in the signature, our treatment has equivalence classes of terms. One could instead take Herbrand's theorem and introduce equality as an uninterpreted relation that satisfies the congruence axioms, and get the same result too.

it follows that $t_i^M = t_i'^M$, and hence $f(t_1, \ldots, t_n)^M = f(t_1', \ldots, t_n')^M$, and hence $f(t_1, \ldots, t_n) \sim f(t_1', \ldots, t_n')$).

Let us rename every element e as the nonempty set $[\![t]\!]$, the equivalence class of t wrt $\sim$, where t is some ground term that evaluates to e in M (such a term must exist since every element in U' is accessible). It is easy to prove that no two elements get named by the same equivalence class. Also, every equivalence class $[\![t]\!]$ is the label of some element in U', namely t^M. We can easily prove, by induction on terms, that that every ground term t evaluates to $[\![t]\!]$ in M'.

It immediately follows that this is a Herbrand model satisfying φ. □

Now, notice that in the proof of Herbrand's theorem, given a model that satisfies a formula, we built the submodel *independent* of the formula. The submodel consisted of all elements accessible using any ground term in the signature. Consequently, the same model construction works in showing that if a *set of universal sentences S* has a model, then it has a Herbrand model as well.

Corollary 6.1 (Herbrand's theorem with equality for sets of formulas) *Let S be a set of universal sentences. Then S is satisfiable iff it is satisfiable in a Herbrand model.*

6.4 Some consequences of Herbrand's theorem

Before we move to completeness, let us observe some simple consquences of Herbrand's theorem. First, it follows that if the signature is countable, then a set of sentences S has a model iff it has a *countable* model. In fact, this is true for any set of formulas as well.

Theorem 6.2 (Downward Löwenheim-Skolem Theorem) *Fix a countable signature Σ. If a set of formulas S over Σ has a model then it has a countable model.*

Proof Every formula in S can be made closed (i.e., made into a sentence) and made universal by Skolemizing it (by replacing variables by new constant symbols and removing existential quantification) to result in equisatisfiable formulas. The resulting set S' and S are equisatisfiable. In fact, it is easy to see that models for S work as models for S', and vice-versa (we can keep the universe, and the interpretation of constants, functions, and relations in the common vocabulary the same). Let S be satisfiable. Then S' is satisfiable as well, and by Herbrand's theorem, there is a Herbrand model for S', which, by definition, is countable. This model can be converted back to a model for S (we keep the same universe, we just throw away the interpretations of the added constants and functions during Skolemization, and interpret variables using the interpretations of constants that replaced them). Hence S has a countable model. □

The above is a surprising result. Every set of axioms A (even infinite ones) that has a model also has a countable model. Recall that there are several complete axiomatizations of theories where the *intended* models are uncountable. For example, there is an axiomatization of reals with addition and multiplication, i.e., for the theory of $(\mathbb{R}, 0, 1, +, \cdot)$. How then do they have a countable model? The only explanation is that even for such theories, there is a countable model that is elementary equivalent (which means it satisfies the same first-order sentences) as the model of reals with addition and multiplication! This is truly bizarre, but true!

There is n generalization of the Downward Löwenheim-Skolem Theorem, called the Löwenheim-Skolem Theorem, which we will not prove in this book, that says that if a formula over a countable signature has a satisfying model that is infinite, then it has models satisfying it of cardinalities κ. In particular, there will always be an uncountable satisfying model. This result is surprising too, as there are complete axiomatizations for certain countable models, like $(\mathbb{N}, 0, 1, +)$. The result then says that this set of axioms also has uncountable satisfying models! These are often referred to as nonstandard models of arithmetic. Again, the key thing is though such models exist, they agree with the standard model on all *first-order expressible properties*.

The above results can also be seen as saying that first-order logic is not powerful enough to talk about infinite cardinalities. A set of first-order sentences can say that the model has at most k elements, for any particular $k \in \mathbb{N}$ $(\exists x_1, \ldots x_k. \forall y \bigvee_{i \in [1,k]} (y = x_i))$. However, there is no set of first-order formulae that ensure that the models that satisfy it are countable, or have any particular cardinality. First-order logic also cannot ensure that satisfying models are *finite*—this is true since validity of first-order logic over finite models is not in r.e., but validity over arbitrary models is in r.e. (as we shall see soon in this chapter).

6.5 Gödel's completeness theorem: FO Validity is recursively enumerable

Let us fix a countable signature Σ.

An instance of the validity problem is a set (finite or infinite, but recursive) A of axioms, which are FO sentences, and a sentence φ. Our goal is to show that the problem of checking validity of such instances, i.e., checking if $A \models \varphi$, is recursively enumerable.

We first negate the formula φ. $A \models \varphi$ iff $A \cup \{neg\varphi\}$ is unsatisfiable. Hence our goal is to prove that $S = A \cup \{\neg\varphi\}$ is unsatisfiable. We convert each formula in S to prenext rectified normal form. We then Skolemize the sentences in S to obtain a set X of sentences over an expanded signature Σ' such that S and X are equisatisfiable. Note that X is itself a recursive set. Our goal is now to show that checking whether X is unsatisfiable is recursively enumerable.

Since X has only universal formulas, we know by Herbrand's theorem that to prove X is unsatisfiable iff X has no satisfying Herbrand model.

Since the signature is countable, the set of all formulas is countable, and hence either X is finite or countable. Let us fix an enumeration of X: $\varphi_1, \varphi_2, \ldots$.

Now any universal formula $\forall \overline{x} \psi$ is true in a Herbrand model iff it is true when the variables $\overline{x}$ are interpreted to be elements corresponding to all possible *ground terms*, since Herbrand models have only interpretations of ground terms in their universe. Consequently, such a universal formula is true in a Herbrand model iff for every tuple of ground terms $\overline{t}$, $\psi[\overline{t} \ / \ \overline{x}]$ holds in the model.

Consequently, it is easy to see that X is satisfied in a Herbrand model iff $\{\psi[\overline{t}] \ / \ \overline{x} \mid \forall \overline{x} \ \psi \in X, t \in GT(\Sigma)\}$ is satisfied in the Herbrand model. This leads us to:

Lemma 6.3 (Term Expansion Lemma) *A set of universal formulas* Γ *is satisfiable iff* $\Gamma^* = \{\psi[\overline{t} \ / \ \overline{x}] \mid \forall \overline{x} \ \psi \in \Gamma, t \in GT(\Sigma)\}$ *is satisfiable.*

Proof If Γ is satisfiable, then clearly Γ^* is satisfied in any model satisfying Γ, and hence is satisfiable as well. Conversely, if Γ^* is satisfiable, then consider a Herbrand model satisfying it (which must exist since the sentences are universal, in fact quantifier-free). Clearly, in this Herbrand model, since all elements are accessible using terms, the formulas in Γ are satisfied as well, and hence Γ is satisfiable. □

Due to the above lemma, we can now take

$$X^* = \{\psi[\overline{t} \ / \ \overline{x} \mid \forall \overline{x} \ \psi \in X, t \in GT(\Sigma)\}$$

and our problem now reduces to showing X^* is unsatisfiable. Note that formulas in X^* are quantifier-free. And quantifier-free formulas admit a decidable satisfiability procedure (see previous chapter). However, even if $A = \emptyset$, X^* can be infinite. Consequently, we cannot subject the X^* to a satisfiability decision procedure.

We now want to show a *compactness theorem* for such quantifier-free sets of formulas. This will allow us to prove unsatisfiability of X^* by just looking at finite subsets of it for unsatisfiability. Note that finite subsets of X^* can be conjuncted and subject to a satisfiability decision procedure, as given in the previous chapter.

Compactness Theorem for quantifier-free grounded formulas

We want to show the following lemma:

Lemma 6.4 *Let* Γ *be a set of quantifier-free grounded sentences. Then* Γ *is satisfiable iff every finite subset of* Γ *is satisfiable.*

Proof If Γ is satisfiable, then, of course, every finite subset of Γ is satisfiable. We hence need to show only the converse.

We will use the propositional compactness theorem to prove this lemma. Note that since every sentence is Γ is grounded, each atomic formula is of the form $t = t'$ or $R(t_1, \ldots, t_n)$, where $t, t', t_1, \ldots, t_n$ are grounded terms.

Let us introduce a set of propositions p_a for every atomic grounded formula a. We can now form a set Γ_p that contains the propositional abstraction of formulas in Γ, obtained by replacing every atomic formula a in any formula in Γ with the proposition p_a.

Now, of course, an arbitrary satisfying assignment of Γ_p may not correspond to a way of satisfying Γ, since equalities obey a set of properties, namely the *congruence axioms* detailed in the last chapter. Let us now introduce a set of propositional constraints that capture these axioms, called Δ.

Δ contains the following formulae:

- $p_{t=t}$ for every $t \in GT(\Sigma)$
- $p_{t=t'} \Rightarrow p_{t'=t}$, for every $t, t' \in GT(\Sigma)$
- $(p_{t_1=t_2} \wedge p_{t_2=t_3}) \Rightarrow p_{t_1=t_3}$, for every $t_1, t_2, t_3 \in GT(\Sigma)$
-
$$\left(\bigwedge_{i \in [1,n]} p_{t_i=t'_i} \right) \Rightarrow \left(p_{R(t_1,\ldots,t_n)} \Leftrightarrow p_{R(t'_1,\ldots,t'_n)} \right)$$
for every relation symbol R of arity n.
-
$$\left(\bigwedge_{i \in [1,n]} p_{t_i=t'_i} \right) \Rightarrow p_{f(t_1,\ldots,t_n)=f(t'_1,\ldots,t'_n)}$$
for every function symbol f of arity n.

It is now easy to show that Γ is satisifiable iff $\Gamma_p \cup \Delta$ is satisfiable. (Proof: If Γ is satisfied in a model M, define a valuation that assigns the propositions p_a to true iff a is true in the model, and argue that Γ_p and Δ will be satisfied under this valuation. Conversely, if $\Gamma_p \cup \Delta$ is satisfied by a propositional valuation, it is easy to see that the equality relation defined by the propositional formulas is a functional congruence over ground terms, and hence defines a Herbrand model of equivalence classes of terms. Interpreting each relation according to the propositional valuation will satisfy the formulas in Γ.

Now, using compactness theorem for propositional logic, we know that $\Gamma_p \cup \Delta$ is satisfiable iff every finite subset of $\Gamma \cup \Delta$ is satisfiable.

Now let us show the required property. If Γ is unsatisfiable, then $\Gamma_p \cup \Delta$ is unsatisfiable, and hence there is a finite subset F of $\Gamma \cup \Delta$ that is satisfiable. Consider $F' = \Gamma_p \cap F$, which is finite. Then the set of FO formulas corresponding to F' in Γ, i.e., the set of formulas whose propositional abstractions are in F', is unsatisfiable (since $F' \cup \Delta$ is unsatisfiable). Hence there is a finite subset of Γ that is unsatisfiable.□

The Algorithm

We now continue and finish our recursively enumerable procedure. Recall that we had left off in showing X^* is unsatisfiable, where X^* was obtained by replacing each universally quantified sentence with all possible instantiations of ground terms.

Using the above lemma, we know that X^* is unsatisfiable iff there is some finite subset of X^* that is unsatisfiable.

We can now build a procedure to find such a finite subset. Recall that for any finite subset of quantifier-free formula, there is a decision procedure (that always halts) whether the set is satisfiable, from the previous chapter. Let's call this decision procedure *DP–QFE* (decision procedure for quantifier-free equality).

6.5.1 The case of finite sets of formulas

We first consider the case when the set of axioms is finite, and hence X is finite. Note that in this case, we can assume the signature is finite too (as the functions/relations not mentioned in the set of formulas clearly do not matter). Note that X is finite, but X^* can be, however, infinite.

Let us consider the following increasing sets that cover X^*. For any $d \in \mathbb{N}$, let T_d denote the ground terms of depth at most d. Formally, these sets are defined recursively as:

- $T_0 = \{c \mid c \ \textit{is a constant symbol in}\ \Sigma\}$
- $T_{d+1} = T_d \cup \{f(t_1, \ldots, t_n) \mid t_1, \ldots, t_n \in T_d, f \ \textit{is a function symbol of arity}\ n\}$

Note that $T_i \subseteq T_j$ for any $i \leq j$, and $\bigcup_{i \in \mathbb{N}} T_i$ is the set of all ground terms.
Our procedure is as follows: Given X, a finite set of universal sentences, we do the following:

```
1. Set $i := 0$;
2. Repeat forever: {
3.    $R := \{\psi[\bar{t} \,/\, \bar{x}] \mid \bar{t}\ \textit{is a tuple of elements in}\ T_i\}$.
4.    Check if $R$ is satisfiable, by calling $\textit{DP-QFE}(R)$.
      If it is not satisfiable, then report $X$ is unsatisfiable and exit (concluding $A \models \varphi$).
5.    Increment $i$;
6. }
```

The correctness of the algorithm is straightforward to see. If $A \models \varphi$, then $A \cup \{\neg\varphi\}$, and hence X would be unsatisfiable. Hence X^* is unsatisfiable. Hence there is a finite subset of $F \subseteq X^*$ that is unsatisfiable. Let FT be the set of terms mentioned in F; then FT is finite. Let i be the maximum depth of the terms in FT. Then in iteration i, the algorithm will instantiate X with all terms of depth i, and hence the set R it constructs will be a superset of F, and hence will be unsatisfiable. Hence the decision procedure call to *DP–QFE* will report unsatisfiable, and the algorithm will halt and report $A \models \varphi$.

On the other hand, if $A \not\models \varphi$, then $A \cup \{\neg\varphi\}$, and hence X would be satisfiable. Hence X^* is satisfiable. In each iteration, the algorithm constructs R which is a subset of X^*, and hence the call to *DP–QFE* will report satisfiable in each round. Hence the algorithm will not halt, and will never declare $A \models \varphi$ holds.

6.5.2 The case for infinite sets of formulas

Let us assume the signature is finite. Let us assume we are asked whether $A \models \varphi$, where A is infinite, but recursive. Again, we know that $A \models \varphi$ iff $A \cup \{\neg\varphi\}$ is unsatisfiable iff the set X constructed by converting formulas in the set to universal formulas is unsatisfiable. This set X is unsatisfiable iff X^* is unsatisfiable. And X^* is unsatisfiable iff there is a finite subset of X^* that is unsatisfiable. The key difficulty is to explore larger and larger finite subsets of X^* systematically such that for every finite subset F of X^*, we eventually will explore a superset of F. There are two infinities to consider here— the set of formulas in X is infinite and the set of terms to instantiate the formulas is also infinite. We need to *dovetail* through the two infinities in order to build our procedure.

Let $En : Y_0, Y_1, Y_2, \ldots$ be an enumeration of certain finite subsets of X^*. Such an enumeration is said to be *fair* if for every finite subset $F \subseteq_{fin} X^*$, there is some $i \in \mathbb{N}$ such that $F \subseteq Y_i$.

There are several ways to achieve a fair enumeration. We give just one example. Consider the enumeration where Y_i consists of the first i sentences in X enumerated by all possible ground terms of depth i. Then clearly this is a fair enumeration. Let F be a finite subset of X^*. Let i be the largest number such that the i'th formula in X, instantiated in some way, belongs to F. Let j be the depth of the largest term that was used to instantiate some element in F. Now, let $k = max(i, j)$. Then it follows that $F \subseteq Y_i$.

For any fair enumeration, we have the following semi-algorithm: Given X, a recursive but infinite set of universal sentences, we do the following:

```
1. Fix a fair enumeration $Y_0, Y_1, \ldots$ of $X^*$
2. Set $i := 0$;
3. Repeat the following forever: {
4.   Check if $Y_i$ is satisfiable, by calling $DP\text{-}QFE(Y_i)$.
     If it is not satisfiable, then report $X$ is unsatisfiable and exit (concluding $A \models \varphi$).
5.   Increment $i$;
6. }
```

Again, the proof that the above algorithm always halts when $A \models \varphi$ and reports that it is so, and the proof that when $A \not\models \varphi$, the algorithm runs forever, is easy to see.

We can extend the above argument also to *countably infinite signatures* and infinite but recursive set of axioms. In this case, we need to dovetail between several infinities— exploring more symbols in the signature, exploring more axioms involving this expanding signature, and systematic term instantiation involving this expanding signature. Again, any fair enumeration will give an r.e. procedure.

6.5.3 Completeness Theorem

We can now phrase our completeness theorem, which follows from the above results.

Theorem 6.3 (Completeness) *Let Σ be a finite or countable signature. Let A be a finite set of sentences or an infinite recursive set of sentences over Σ, and let φ be a sentence over Σ. Then the problem of checking whether $A \models \varphi$, is recursively enumerable.*

Let us now work out an example.

Example 6.2 Consider the group axioms, where we have a special constant e for the identity element:

- Associativity: $\forall x, y, z.\ f(f(x, y), z) = f(x, f(y, z))$
- Identity: $\forall x. f(x, e) = x \wedge f(e, x) = x$
- Inverse: $\forall x \exists y. f(x, y) = e \wedge f(y, x) = e$

Let us now take the above three sentences as the set of axioms A. And let us try to prove the following formula, which says the identity is unique, i.e.,

$$\varphi : \forall e'.\ ((\forall x.(f(x, e') = x \wedge f(e', x) = x) \Rightarrow (e = e')))$$

Of course, the above property is true even of monoids, i.e., even when the first two axioms hold. However, let's consider all group axioms for this example.

The first two formulas are already in prenex rectified normal form and universal. Skolemizing the third axiom using a new function g gives:

$$\forall x f(x, g(x)) = e \wedge f(g(x), x) = e$$

The new function symbol g intuitively corresponds to a function that provides the inverse of an element. (We don't need to know it is unique in order to ask that such a function exists.)

The formula φ is not in prenex form; bringing it to prenex form gives:

$$\begin{aligned}\varphi &\equiv \forall e'.\ (\neg(\forall x.(f(x, e') = x \wedge f(e', x) = x) \vee (e = e'))) \\ &\equiv \forall e'.\ ((\exists x.(\neg(f(x, e') = x) \vee \neg(f(e', x) = x)) \vee (e = e'))) \\ &\equiv \forall e'.\ \exists x.(\neg(f(x, e') = x) \vee \neg(f(e', x) = x) \vee (e = e'))\end{aligned}$$

The negation of φ is hence:

$$\neg\varphi \equiv \exists e'.\ \forall x.(f(x, e') = x \wedge f(e', x) = x \wedge \neg(e = e'))$$

Skolemizing the above, by replacing the quantified variable e' by a new constant symbol c gives:

$$\forall x.(f(x, c) = x \wedge f(c, x) = x \wedge \neg(e = c))$$

We now have a set X containing four universal formulas:

- $\forall x, y, z.\ f(f(x, y), z) = f(x, f(y, z))$
- $\forall x. f(x, e) = x \wedge f(e, x) = x$
- $\forall x f(x, g(x)) = e \wedge f(g(x), x) = e$
- $\forall x.(f(x, c) = x \wedge f(c, x) = x \wedge \neg(e = c))$

And our task is to check whether they are simultaneously satisfiable.

Let us instantiate with the depth 0 ground terms, i.e., by constants e and c. Then we get the formulae where all quantified variables are replaced by all possible combinations of e and c. That's 14 quantifier-free formulas!

Note that this set includes the following formulae:

- The second formula with x replaced by c:
  $f(c, e) = c \wedge f(e, c) = c$
- The fourth formula with x replaced by e:
  $(f(e, c) = e \wedge f(c, e) = e \wedge \neg(e = c))$

Clearly these two formulas are not satisfiable in any model. If $f(c, e) = c$ and $f(c, e) = e$, then we must have $c = e$, which contradicts the conjunction $\neg(e = c)$.

Hence when we ask the decision procedure for quantifier-free formulae whether the 14 formulas have a model, it will report unsatisfiable, and the algorithm above would conclude $A \models \varphi$.

We invite the reader to in fact generate the above formulae, and give them to an SMT solver, like Z3 or CVC4, in order to check that the quantifier-free formulae are unsatisfiable.

Example 6.3 We can take the same axioms above, and try to show that the following holds, which says that inverses are unique. Since we have used the function g, during Skolemization of the axioms, to give us the inverse of elements, let's use the same function g (for brevity).

$$\varphi : \forall x, y.\ (f(x, y) = e \wedge f(y, x) = e) \Rightarrow (y = g(x))$$

Negating the above and Skolemizing using two new constant symbols c and d gives:

$$(f(c, d) = e \wedge f(d, c) = e) \wedge \neg(d = g(c))$$

Instantiating the Skolemized axioms and the above formula with depth 0 terms (i.e., by the constants e,c, and d) gives a large set of quantifier-free formulae, and it turns out that they are already unsatisfiable. We encourage the reader to write these formulae and feed it to an SMT solver to check that this is indeed so. Consequently, $A \models \varphi$.

6.6 Observations and Consequences

Using SMT solvers:

The above presentation was carefully done so that we get an r.e. procedure that repeatedly calls a solver to check satisfiability of quantifier-free formulae with equality. One can instead also go all the way down to propositional logic satisfiability, and implement the satisfiability of quantifier-free formulae using satisfiability of a propositional encoding of it. This was in fact proposed by Gilmore in 1960! Since SMT solvers already implement satisfiability of quantifier-free formulae with equality, and avoids the blow-up that the propositional encoding entails, we prefer this technique. Furthermore, we will see another application of this term instantiation in a later chapter that allows us to *combine* quantified theories.

The Bernays-Schönfinkel-Ramsey/EPR class

Let us now consider a signature without any function symbols, and a finite set S of formulas of the form $\exists\overline{x}\forall\overline{y}\varphi$. We are asked to check if S is satisfiable. Skolemizing these formulas could introduce new constants but *no new functions*. Consequently, we end up with a set of universal formulas X that we need to check for satisfiability. Since there are no function symbols, the only ground terms are the constants, and we can assume that the constants are only those that occur in the formula, without loss of generality. Consequently, the r.e. procedure outlined earlier in this section can stop after the first instantiation of constants! Hence it is a decision procedure (which always halts on all inputs) and decides satisfiability of such formulae. This fragment of FO formulae, namely $\exists^*\forall^*$ sentences over a signature that has no function symbols, hence admits a decidable satisfiability problem, and is called the *Bernays–Schönfinkel-Ramsey* class or the *effectively propositional reasoning* (EPR) class. Note that for validity, the fragment that is decidable is the $\forall^*\exists^*$ fragment where the signature has no function symbols. This is one of the few quantified fragments of first-order logic that admits decidable validity.

Decidability when Axioms are Negation Complete

A set of axioms A is said to be *consistent* (without contradiction) if there is no sentence such that $A \models \varphi$ and $A \models \neg\varphi$, i.e., $\varphi, \neg\varphi \in Th(A)$. Note that a set of axioms A is consistent iff there is at least one model satisfying the axioms A.

A set of axioms A is said to be *complete* (or *negation complete*) if for every sentence φ, either $A \models \varphi$ or $A \models \neg\varphi$. In other words, the theory of A, $Th(A)$, contains either φ or $\neg\varphi$.

For example, the set of axioms of Presburger arithmetic is consistent and complete. The set of axioms of groups is consistent but not complete.

A consequence of the results of this section is that the theory any complete and consistent axiomatizations is *decidable*. Given a sentence φ, we can execute two copies of the r.e. procedure defined in this section to check whether $A \models \varphi$ and whether $A \models \neg\varphi$. These two executions must be simulated essentially in parallel— for example, running one procedure k steps and then switching to the other for k steps, and then switching back, forever, for some fixed k. Since either $A \models \varphi$ or $A \models \neg\varphi$, one of these procedures will terminate, in which we can halt, and report whether φ is in the theory or not.

Theorem 6.4 *Let A be a recursive set of sentences that is complete. Then the theory of A, $Th(A)$ is decidable.*

The above also means that if the theory of a single structure is *undecidable*, then it is not axiomatizable. We will prove (see next chapter) that the theory of $(\mathbb{N}, 0, 1, +, \times)$ is undecidable. This means that there is *no recursive set of FO axioms A* such that the theory of A is identical to the theory of this model! This is in fact a version of Gödel's first incompleteness theorem.

Axiomatizability and recursive enumerability

We proved completeness for any recursive set of axioms. However, it is easy to extend the result even when the axioms are recursively enumerable— the procedure will enumerate axioms and instantiate them systematically.

Consider a class of structures C. The notions of having a recursively enumerable set of axioms A that characterize the theory (i.e., $Th(A) = Th(C$ and having $Th(C)$ itself being recursively enumerable are synonymous. If a r.e. set of axioms A exists characterizing C, then by the completeness theorem, $Th(\mathcal{A})$ is r.e. as well. On the other hand, if $Th(C)$ is r.e., then we can choose as axioms this theory itself.

Axiomatic Systems

The most important consequence of the completeness theorem is that it justifies the axiomatic approach. We are typically interested in logic over a particular single structure, or interested in a class of structures. There are many ways to define such a single structure or a class of structures, even using finite means (for example, we can define them using computable functions— giving functions that decide which strings over an alphabet are the elements of a univers, and providing programs that oeprationally define functions and relations). The axiomatic method, in contrast, asks the class of structures to be defined using properties which are themselves written in FOL. And the completeness theorem gives the guarantee that validity of such a set of axioms is always r.e., which roughly means that every theorem has a proof.

Compactness Theorem for FOL

Another consequence of the results of this section is that the compactness theorem holds for first-order logic sentences as well.

Theorem 6.5 *Let Γ be a set of first-order sentences over a countable signature Σ. Γ is satisfiable iff every finite subset of Γ is satisfiable.*

Proof The forward direction is trivial. For the converse, assume Γ is unsatisfiable. Then, by the results of this section, we can assume Γ is a set of universal sentences. Then, by Lemma 6.3 (Term Expansion Lemma), $\Gamma^* = \{\psi[\bar{t}/\bar{x} \mid \forall x\psi \in \Gamma, t \in GT(\Sigma)\}$ is unsatisfiable. In other words, the set of quantifier-free sentences obtained by instantiating variables by all possible ground terms is unsatisfiable. By Lemma 6.4, there exists a finite subset F^* of Γ^* that is unsatisfiable. Let $F \subseteq \Gamma$ be a finite subset of Γ from which the elements of F^* were obtained (using term instantiation). Then F is unsatisfiable as well (since even instantiations of variables by ground terms make it unsatisfiable). Hence there is a finite subset of Γ that is unsatisfiable. □

Chapter 7
Number Theory and Correctness of Programs: Incompleteness

Natural numbers, endowed with addition and multiplication, is one of the most ubiquitous structures in mathematics and our society. The reason it's so useful is that it can model so many things in our world, both the physical/natural world and the human created world, especially those that involve *discrete* objects. If I was a single lone jellyfish in deep sea where everything was murky and there was nothing discrete to discern, I may have less use for natural numbers. But we live in a world that is full of discrete objects and others that can be discretized by approximating them. For example, people are discrete objects; it's useful to know how many children one has, or how many people can vote in a country. Goats and cows are discrete, and important in early notions of wealth and trade. Time is not discrete, but we can discretize time into intervals, like seconds, and hence use numbers to count time. Planetary positions are not discrete, but can be discretized to arcs of degree, and hence modeled as natural numbers. With discretization, a significant aspect of the physical world can be modeled as numbers, and most *observations* of physical phenomena can be modeled using numbers.

The theory of numbers is hence ubiquitous and studied extremely well in mathematics. We learned how to represent them succinctly and how to do operations on them (using n-ary representations, which need a symbol for 0, and algorithms for computing operations on them). Much of elementary school is devoted to learning these simple algorithms.

Programs are also very much discrete in nature. They deal with inputs that are sequences, which can be seen as numbers, they do operations, which can be seen as similar to operations involving numbers (arithmetic/Boolean circuits), and the program itself, is a sequence of symbols that can be seen as numbers.

Consider an imperative program (choose your favorite programming language) with *assertions*. Assertions are basically properties of states that you assert in code, and are a form of *specification* asserting that the property must be true in all states where the assertion is reached. A program with assertions is said to be *correct* if in every execution of the program, whenever an assertion is reached, the asserted property holds.

The problem of *program verification* is to determine whether a given program with assertions is correct. The statement that a program P with assertions is correct, is really a theorem in mathematics. And a *proof* of such a theorem, no matter what the notion of proofs are, is a mechanically checkable sequence of statements, where it should be clear that the proof asserts in the end that the program is correct. There are several proof systems that prove programs correct— *Hoare logic* is a popular one. But you can imagine other formal arguments/proofs of why a program satisfies a particular assertion.

Incompleteness results in logic argue that there are no formal systems that can prove all theorems in certain models or classes of models. In other words, for certain models or classes of models, *not all theorems have proofs, in any proof system*.

In my view, there are at least *three* incompleteness results that are astonishing: (a) theorems expressed in FOL over natural numbers with addition and multiplication (Gödel's incompleteness theorem), (b) theorems expressed in FOL about the class of all finite structures (or even all finite graphs), and (c) theorems about correctness of programs.

In this book, we show that none of the above problems is r.e., which is another way of saying there is no formal proof system that contains proofs for these theorems. The incompleteness of all three problems are non-intuitive— it seems intuitive that all FO theorems about numbers ought to have proofs, that all FO properties of finite graphs ought to be provable, and all correct programs should have some proof of correctness. It's incredible that they don't. We intrinsically believe that all theorems ought to be provable, but incompleteness argues that this is not possible, at least not in any formal proof system. It shows that formal systems are intrinsically weak!

Incompleteness results put mathematics in a strange place than we intuitively imagined. There may be true theorems that may not even be provable using the formal rules of proof we accept. It opens up the possibility that theorems in number theory (including open problems) may not even have proofs. And similarly theorems about finite graphs. And similarly, the correctness of certain programs.

We proved the incompleteness of FO theorems of finite structures in Chapter 4 (Trakhtenbrot's theorem). We will show the other two incompleteness results in this chapter.

The incompleteness results all have a similar proof outline based on some form of *diagonalization*, similar to the one found by Cantor, and similar to the one used by Turing to show the undecidability of the halting problem. Since, as computer scientists, we already know of such results, in particular that the halting problem for Turing machines (or programs) is undecidable and the *non-halting* problem of Turing machines (or programs) is not even recursively enumerable, we will use these to prove our results.

7.1 Program Verification

Consider a TM that we want to check for non-halting. The TM can be realized by a program P (any programming language with infinite memory will do, as it can simulate the moves of a Turing machine; for example, a program with access to unbounded linked lists, or a program with access to an unbounded secondary storage device, or even a program that has unbounded integers). So the problem reduces to checking whether P does not halt.

Construct a program P' that is basically the program P modified so that if P halts, we add an assertion `assert false;` at the point where it halts. (An assertion of *false* doesn't hold in any program state— if you are uncomfortable with it, replace it with $x := 1; assertx = 0;$.)

Now it is clear that the program P' satisfies its assertion if and only if P does *not* halt. Consequently, program verification is not recursively enumerable.

Theorem 7.1 *Program verification is not a recursively enumerable problem.*

Let us now use the above result to prove Gödel's incompleteness theorem stated in terms of the theory of numbers not being recursively enumerable.

7.2 Incompleteness of the theory of natural numbers with additional and multiplication

We want to show that the theory of natural numbers with addition and multiplication is not recursively enumerable, i.e., $Th(\mathbb{N}, 0, 1, +, \times, =, \leq)$ is not recursively enumerable.

The intuition behind this result is that the theorem stating that a program is correct (or that Turing machines halts or does not halt) *is a first-order expressible theorem in number theory*! Consequently, there is no formal proof system such that all theorems in FO arithmetic have proofs in the system.

We want to reduce the problem of Turing machine non-halting (or halting[1]) to the validity problem of sentences over the theory of natural numbers. The following proof is adapted from Dexter Kozen's book "Automata and Computability".

First, we can express several interesting properties using first-order logic using addition and multiplication:

- q is the quotient and y is the remainder when x is divided by y:

$$IntDiv(x, y, q, r) : \;\; x = qy + r \wedge r < y$$

- y divides x:

$$Div(y, x) : \;\; \exists q.\, IntDiv(x, y, q, 0)$$

[1] Note that reducing the halting problem to validity also works to show non r.e.-ness since the theory is negation-complete; a negation conmplete theory is either decidable or not r.e.

- x is prime:

$$Prime(x): \quad x \geq 2 \wedge \forall y.\ (Div(y,x) \Rightarrow (y = 1 \vee y = x))$$

- y is a power of a particular fixed prime p, i.e., $y = p^k$ for some $k \in \mathbb{N}$:

$$Power_p(y): \quad \forall z.((Div(z,y) \wedge Prime(z)) \Rightarrow z = p)$$

We will now show a reduction from the non-halting problem of a Turing machine (on an empty tape) to validity of arithmetic sentences.

Given a Turing machine M with tape alphabet Γ and states Q, let us fix the alphabet $\Pi = \Gamma \cup (Q \times \Gamma)$. Let us choose a prime p larger than Π, and let us look upon sequences over Π as p-ary representations of numbers. A sequence $a_n \ldots, a_0$, where each $a_i \in [0, p-1]$ maps to the number $\Sigma_{i \in [1,n]}\ a_i p^i$.

The *computation* of M on the empty tape can be seen as a sequence of configurations $\sigma_0, \sigma_1 \ldots,$, where each σ is a configuration represented a word in Π^*. When M halts, configurations are bounded by some maximum length (depending on how much space M takes on the tape. Let H denote the subset of Π that has the halting state: $H = \Gamma \times HQ$, where $HQ \subseteq Q$ are the halting states. Hence M halts iff there is a sequence of configurations that represents valid moves of M that has the halting configuration, i.e., where some element of H occurs.

We now encode the halting of M as the existence of a number whose p-ary representation encodes a valid halting computation of M.

A finite computation sequence $\sigma_0, \sigma_1, \ldots, \sigma_n$ will be encoded as large enough blocks so that each σ_i fits into a block. If C is a large enough length to encode each configuration, we will use $c = p^C$ to capture this number. (In general, most numbers k related to the Turind machine that we need will be captured using p^k instead of k.) This number c will eventually be quantified in the formula we reduce to.

In order to say that a configuration sequence is correct, it is sufficient to demand that successive configurations are correct. In order to demand $\sigma, sigma'$, two successive configurations (encoded with sequence of the same length C) are correct, it is sufficient to check every *three-element* subsequence of σ with the corresponding three-element subsequence in σ' (since Turing machines make only local changes on the tape). The three element sequences either does not encode a state (i.e., is over Γ only), in which they must be the same, or the three element sequence in σ encodes a state in the middle, in which case the corresponding three-element sequence in σ' depicts the correct evolution according to the transitions of the Turing machine.

Let V be the set of all 6-tuples $(a_1, a_2, a_3, b_1, b_2, b_3)$ that denote valid pairs of three-tuples. V includes all:

- Every $(a_1, a_2, a_3, b_1, b_2, b_3)$ such that $a_1, a_2, a_3, b_1, b_2, b_3 \in \Gamma$
- For every transition $\delta(q,a) = (b, q', R)$, the triples $(a_1, (q,a), a_2, a_1, b, (q', a_2))$, $((q,a), a_1, a_2, b, (q', a_1), a_2)$, and $(a_1, a_2, (q,a), a_1, a_2, b)$ are in V, for every $a_1, a_2 \in \Gamma$.
- For every transition $\delta(q,a) = (b, q', L)$, the triples $(a_1, (q,a), a_2, (q', a_1), b, a_2)$, $(a_1, (q,a), a_3, (q', a_1), b, a_3)$,

are in V, for every $a_1, a_3 \in \Gamma$.

Note that check inconsistencies of sequences only in tuples where the "middle" symbol in the first configuration, i.e., a_2, encodes a state.

We will write a formula that ensures that in a number encoding sequences of configurations, for every two consecutive configuration σ and σ', every three-element subsequence in σ and the corresponding three-element subsequence in σ', the 6 elements are related by V. This will ensure that the entire sequence of configurations is valid.

The crucial power of arithmetic with addition and multiplication is that we can encode sequences as numbers, and also *decode* sequences into their components. Here is an important formula, which says that the character in position Y of a sequence encoded by the number v is a (where $a \in [0, p-1]$. As we said before, we encode the position Y using the number $y = p^Y$. So the following really says that the position encoded in y of the sequence encoded by v is b (assuming y is a power of p):

$$Digit(v, y, a) = \exists u.\ \exists r.\ (v = r + ay + upy \wedge r < y \wedge a < p)$$

Intuitively, let's say v's p-ary representation can be split into $\rho_1 \cdot a \cdot \rho_2$, where $|\rho_2| = Y$. Then clearly $v = r + ap^Y + up^{Y+1}$, for some $r < y$ (where r encodes the number corresponding to ρ_2 and u encodes the number corresponding to ρ_1). Replacing p^Y by y gives $v = r + ay + upy$, which is what the above formula demands.

The intuition for the following formulae follow along similar lines, and we let the reader work this out for themselves.

We can demand that the 3-digit sequence of v at positions encoded by y are b_1,b_2, and b_3, using the formula:

$$3Digit(v, y, b_1, b_2, b_3) : \exists u.\ \exists r.\ (v = r + b_1 y + b_2 py + b_3 ppy + upppy$$

$$\wedge\ r < y\ \wedge\ b_1 < p\ \wedge\ b_2 < p\ \wedge\ b_3 < p)$$

Now we can demand that the three digits of v at the position encoded by y match correctly the three digits of v at the position encoded by z:

$$Match(v, y, z) : \bigvee_{(a_1,a_2,a_3,b_1,b_2,b_3)\in V} (3Digit(v, y, a_1, a_2, a_3) \wedge 3Digit(v, z, b_1, b_2, b_3))$$

We can now write a formula that says that the p-ary string that represents v encodes a valid sequence of configurations of the TM evolution. For technical reasons, we will parameterize this with c and d— c is the number that encodes the size of configurations (i.e., p raised to the power of the length of configurations) and d will encode a bound on the entire length of the sequence v. The formula checks whether all pairs of three-digit sequences precisely c apart (or rather $log_p(c)$ apart) in v match according to the Turing machine's moves, up to d:

$$ValidMoves(v, c, d) : \forall y.(Power_p(y) \wedge yppc < d) \Rightarrow Match(v, y, yc)$$

We can state the sequence representing v starts with the initial configuration. Let *init* be the number encoding the symbol $(q_0, \#)$, the Turing machine reading the blank symbol. Let *blank* denote the number encoding the blank symbol #. Note that the start configuration is then $(q_0, \#) \cdot \# \cdot \# \ldots \cdot \#$. The following formula forces this as the first configuration of v:

$$\mathit{Start}(v, c) : \mathit{Digit}(v, 1, \mathit{init}) \wedge \forall y. \big(\mathit{Power}_p(y) \wedge y > 1 \wedge y < c \Rightarrow \mathit{Digit}(v, y, \mathit{blank})\big)$$

We can also state that the halting configuration occurs in v before d by the formula:

$$Halt(v, d) : \exists y. \left(\mathit{Power}_p(y) \wedge y < d \wedge \bigvee_{b \in H} \mathit{Digit}(v, y, b)\right)$$

We can now ready to write a formula that says that v is a valid sequence of configurations that halts. In order to do this, we first express that the number d represents an upper bound on the length of v (we then interpret v using the p-ary representation of v, with 0's padded to the left, if necessary):

$$\mathit{Length}(v, d) : \mathit{Power}_p(d) \wedge v < d$$

Note that v along with d (where $\mathit{Length}(v, d)$ holds) represents the precise p-ary sequence we wish to express properties about. We can now write that this sequence represents a valid halting computation:

$$\mathit{ValidHaltComp}(v) : \exists d. \exists c. \ (\mathit{Length}(v, d) \wedge \mathit{Power}_p(c) \wedge \ c < d$$

$$\wedge \mathit{Start}(v, c) \wedge \mathit{ValidMoves}(v, c, d) \wedge \mathit{Halt}(v, d))$$

We can now finally write a sentence that says that the Turing machine M does *not* halt:

$$\neg \exists v. \ \mathit{ValidHaltComp}(v)$$

The above formula is valid over the standard model of natural numbers with addition and multiplication iff the Turing machine does not halt. The relation < can be expressed with the other relations using the following equivalence:

$$x < y \Leftrightarrow \exists z. \neg(z = 0) \wedge x + z = y$$

We hence have:

Theorem 7.2 *The first-order theory of natural numbers with addition and multiplication, $Th((\mathbb{N}, 0, 1, +, \times, =))$ is not recursively enumerable.*

7.3 Further Remarks

Gödel's proof and strengthenings

The crux of the above proof is that sequences over an alphabet, related in simple syntactic ways (like the moves of a Turing machine) can be encoded in arithmetic. Gödel was the first to discover this, and he used it in what's called Gödel numbering in order to encode *proofs* into numbers; proofs are also sequences whose validity is syntactic. This was done before a solid notion of computation (such as Turing machines or lambda calculus) existed. In fact, Gödel showed that one can encode a self-referential formula, where for any reasonable proof system, one can state in arithmetic a statement that says: "There is no proof of this statement." A proof system is damned it it proves this statement, and damned if it does not. If it proves it, then it's proved a wrong statement! And if it doesn't prove it, it's a correct statement it cannot prove! Gödel's proof combines encoding proofs/sequences as numbers and a diagonalization argument. In our proofs, we proved undecidability and hence non-r.e.-ness of Turing machine non-halting using diagonalization, and a separate proof of encoding existence of sequences into statements about numbers.

Note that the above proof extends beyond first-order arithmetic. Any logic that is more powerful than first-order arithmetic does not have a complete proof system. In fact, it turns out that the above theorem can be strengthened to show that even *quantifier-free arithmetic with addition and multiplication* (i.e., implicitly universally quantified) is undecidable and not recursively enumerable. In fact, the even simpler problem of solving Diophantine equations (given a set of polynomial equations, deciding whether there is solution using integer values) is undecidable, and checking whether there is no solution to them is not r.e.. This is a celebrated problem, called Hilbert's Tenth Problem, which was open for a long time and settled in a famous theorem by Yuri Matiyasevich in 1970.

Axiomatizations

Due to the *completeness theorem*, we know that any model whose theory is axiomatizable is *decidable*. Since the theory of arithmetic with addition and multiplication is undecidable, it follows that there can be no recursive axiomatization of it.

The Peano axioms formulated in first-order logic was an attempt to axiomatize arithmetic. It has an *infinite* set of axioms, including an axiom schema for induction, which essentially says that any first-order property about numbers (formulated as a formula with a single free variable) can be proved by induction. However, as we know from the results in this section *and* the completeness theorem, this axiom system, if sound, must be incomplete. In fact, there are natural *concrete* theorems that can be stated in FOL that are not provable in Peano arithmetic (see the results of Paris and Harrington where a version of Ramsey's theorem is shown to be unprovable in

Peano arithmetic). The *Principia Mathematica* is another formal system for which the incompleteness theorem applies, showing that there can be no recursive of it that is consistent and complete.

Returning to Program Verification: The Method using Invariants

Consider the problem of program verification again. How do people actually prove programs correct (partially correct, i.e., satisfy their assertions) in practice? The predominant method is the *invariant* method, which is basically a proof by induction. People postulate essentially a set of configurations $Inv(\overline{x})$ (called an invariant), captured as a formula in logic over a set of variables $\overline{x}$, and prove the following properties about it:

- The initial states are contained in *Inv*:

$$\forall \overline{x}.Init(\overline{x}) \Rightarrow Inv(\overline{x})$$

- If $Post(\overline{x}, \overline{x}')$ represents how the program can change configurations in a single step, then the invariant is closed under *Post*:

$$\forall \overline{x}, \overline{x}' : (Inv(\overline{x}) \wedge Post(\overline{x}, \overline{x}')) \Rightarrow Inv(\overline{x}')$$

- The invariant set and the set of *unsafe* states where the assertion is violated, do not intersect:

$$\forall \overline{x}(Inv(\overline{x}) \Rightarrow \neg Unsafe(\overline{x}))$$

Once we postulate such an invariant set, program verification boils down to proving validity of the above assertions! Clearly, if there is such an invariant set that contains all the initial states and closed under any move done by the program, then the set contains all the reachable states, and since it doesn't intersect the unsafe sets, it satisfies the assertion.

Verification methodologies such as that of Floyd and Hoare are essentially stylized proof techniques that follow the above method (expressing invariants at only loop headers or method boundaries). In fact, these stylistic methods break down (become too weak) for complex programs (such as concurrent programs or programs that pass programs/program-pointers as parameters); however, the global invariant method above is still robust and always is viable.

Now, one could ask whether such an invariant always exists. It clearly does whenever the program is correct— choose the invariant to be the set of *all* reachable states of the program: it satisfies all the above requirements.

So then where exactly lies the problem in not being able to prove a program correct? It lies in two aspects: (a) invariants may exists but may not be *expressible* in logic, and (b) invariants may be expressible in logic, but there may be no *proofs* (in a fixed formal system) that the above formulas are valid, i.e., the logic used to express the above conditions may be incomplete.

It turns out that either can happen. If we choose a weak enough logic, say a decidable logic, then we would be able to decide validity of the above formulas... but the invariant may not be expressible in logic. However, it turns out that for any reasonable programming language, the *invariant* (or the precise set of reachable states) is *always* expressible in a powerful enough logic, such as arithmetic with addition and multiplication! However, then, the logic becomes incomplete, and there may be no proofs for proving the above properties. And hence program verification, in general, remains incomplete either way. In automated program verification, one either chooses a weaker logic and builds automated decision procedures to check the properties above (this is good for shallow properties), or chooses a very expressive logic, and builds incomplete automation for logical reasoning to find proofs that establish the above properties!