Chapter 7:
Predicate Logic: Syntax, Semantics, Undecidability

This chapter summarizes parts of chapters 7, 8, and 9 of the textbook, and touches upon the subject of chapter 12.

Many applications require a more expressive language than that of propositional logic. A major one is mathematics itself: from the beginning, mathematical logic was formulated to allow for the use of predicate symbols (to denote sets and relations); function symbols; and variables ranging over some domain, together with existential and universal quantification over those variables. Logics equipped with such features are referred to as predicate logic, or alternatively as first-order logic (so-called "higher-order logics," such as second-order logic, allow quantification not only over elements of the domain, but over subsets of the domain). Similar features are needed for computer-engineering applications such as databases, program-correctness proofs, and artificial intelligence.

This chapter will review the syntax and semantics of predicate logic, and briefly present a Hilbert-style deductive system. But, having just covered SAT solvers, in addition to propositional resolution, we first briefly discuss an important price that is paid for the expressivity of predicate logic: the failure of decidability.

Undecidability of predicate logic

As has already been mentioned, the great German mathematician David Hilbert essentially envisioned the automation of mathematics. Given that mathematical questions could be formulated in first-order logic, and that proofs only required adherence to certain formal rules, could mathematical questions be answered by algorithms? Hilbert's program played an important role in organizing mathematical research throughout the twentieth century, despite being shown early in the decades to be doomed to failure.

One reason, among others, is that satisfiability, and therefore, validity, is not *decidable* in predicate logic: there is no algorithm that, when presented with an arbitrary formula, will always terminate and correctly determine whether or not that formula is satisfiable. This result follows from fundamental limitations on computation discovered by the British computer scientist

Alan Turing.

Turing's results specifically identify fundamental limitations of formal verification of computer systems: they relate to the problem of deciding whether an algorithm will terminate when run on a given input. For this reason, they are known as aspects of the *halting problem*.

The halting problem

These results require a formal definition of an algorithm. However, Turing's key idea can be understood informally. It requires us to think about programs whose inputs are themselves programs: but naturally, any program that performs software verification is of just that sort. Consider therefore the problem of deciding whether a given program P terminates – or halts – in the reflexive scenario where its input is P itself.

Turing's proof proceeds by contradiction. Suppose that there exists a program P^* that solves the problem: when presented with an arbitrary program P as its input, it eventually halts, and determines whether P halts when run on its "own" input, P. If there did exist such a program P^*, a minor modification could be made: in cases where it was determined that the input program P did halt on its own input, it could be made to go into an infinite loop, and never terminate; otherwise, it could simply terminate normally.

Call the modified program P^{**}. What would happen if this program were run on its own input? If it determined that P^{**} terminated on its own input, then P^{**} would go into an infinite loop, and never terminate; on the other hand, if P^{**} did not terminate on its own input, it would terminate – on its own input. This contradiction shows that the assumption of the existence of P^* must be incorrect: there can exist no program that determines whether an arbitrary program halts on its own input. The 'halting problem,' however artificial, is therefore an example of a problem that is undecidable.

The halting problem therefore proves the existence of problems that are algorithmically undecidable. But it also furnishes a means of proving other problems undecidable. For instance, there are other versions of the halting problem, such as that of deciding whether an arbitrary program P halts on the 'empty input.' Any program P can be modified so that, if run on the empty input, it first feeds itself its own input P. Call the modified version P'; then P' halts on the empty input if and only if P halts on its own input. This means that the first version of the halting problem 'reduces' to that

of deciding whether an arbitrary program P' halts on the empty input. If it were possible to decide whether the latter occurs, it would be possible to decide whether an arbitrary program P halts on its own input; but Turing showed that to be impossible. So there is no algorithm that decides whether an arbitrary program P halts on the empty input.

More generally, any decision problem to which an undecidable problem reduces must itself be undecidable. Such is the case for satisfiability of predicate-logic formulas. Given an arbitrary program P, one can write down a predicate-logic formula that is satisfied if and only if, say, P halts on the empty input. Proofs of the 'undecidability of predicate logic' are generally based on a reduction from another undecidable problem, such as the halting problem for 'two-counter machines' described in Chapter 12 of the textbook. But the general idea is that the language of predicate logic can express the halting of an arbitrary program on, say, the empty input, and the satisfiability of an arbitrary formula is therefore undecidable.

There does exist a version of resolution for predicate logic. Moreover, it is both sound and complete. But the undecidability of predicate logic manifests itself in the fact that searches for first-order resolution refutations are not guaranteed to terminate.

Though Hilbert's program of automating mathematics was doomed by results such as Turing's, it led to fundamental discoveries in mathematics and in formal methods of mathematical reasoning. Despite the discovery that such formal methods have important limitations – particularly when applied to the analysis of algorithms – predicate logic remains the foundation of mathematics, and 'automated reasoning' continues to grow in importance as a means of design and analysis of computer systems.

7.2 Formulas in first-order logic

Definition 7.6 Let $\mathscr{P}$, $\mathscr{A}$, and $\mathscr{V}$ be countable sets of *predicate symbols*, *constant symbols*, and *variables*. Each predicate symbol $p^n \in P$ has an *arity*, the number $n \geq 1$ of *arguments* that it takes. If $n = 1$, the predicate symbol is *unary*; if $n = 2$, it is *binary*; in general, it is *n-ary*. The arity will usually not be indicated explicitly; it will be clear from the number of arguments employed.

We shall introduce *function symbols* later. The symbols listed above are used to define an important fragment of first-order logic – one that is used, for example, in querying relational databases.

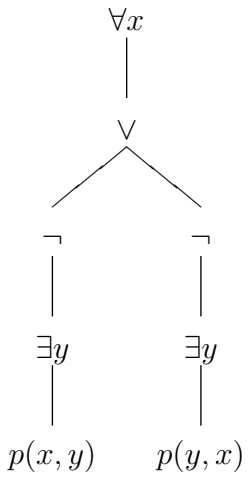
Definition 7.7 $\forall$ is the *universal quantifier*, read, 'for all,' and $\exists$ is the *existential quantifier*, read, 'there exists.'

Definition 7.8 An *atomic formula* is an n-ary predicate symbol followed by a list of n arguments in parentheses: $p(t_1, t_2, \ \ldots \ , t_n)$, where each t_i is either a constant symbol or a variable. More generally, a *well-formed formula (wff)*, or simply, a *formula*, of first-order logic is a tree defined recursively as follows:

- a leaf labelled by an atomic formula is a formula;
- a node labelled by $\neg$ with a single child that is a formula is a formula;
- a node labelled by $\forall x$ or $\exists x$, for some variable x, with a single child that is a formula is a formula;
- a node labelled by a binary Boolean operator with two children that are formulas is a formula;
- the set of all first-order formulas is the smallest set that satisfies the above conditions.

The concept of structural induction carries over straightforwardly from the propositional case. When first-order formulas are written as strings, the quantifiers have the same level of precedence as negation, and a higher level than those of the binary Boolean operators.

Example 7.9 The formula shown below can be represented in string form as $\forall x \ (\neg\exists y \ p(x, y) \vee \neg\exists y \ p(y, x))$.



□

7.2.2 The scope of variables

Definition 7.11 The *scope* of the quantifier $\forall x$ in the quantified formula $\forall x\ A$ is the formula A. It is not necessary for x actually to occur within A.

- In the above example, the scope of the universal quantifier $\forall x$ is $\neg\exists y\ p(x, y) \vee \neg\exists y\ p(y, x)$;
- in the first existentially quantified formula above, the scope of the existential quantifier $\exists y$ is $p(x, y)$;
- in the second existentially quantified formula, the scope of $\exists y$ is $p(y, x)$.

Definition 7.12 Let A be a formula. An occurrence of a variable x in A is a *free* occurrence if it does not lie within the scope of a quantifier $\forall x$ or $\exists x$. An occurrence that does lie within the scope of such a quantifier is a *bound* occurrence. A variable may have both free and bound occurrences within the same formula. A *free variable* of a formula is one that has at least one free occurrence in the formula.

If a formula has no free variables, it is *closed*. If $x_1, x_2, \ldots, x_n$ are all of the free variables of A, then the *universal closure* of A is $\forall x_1 \forall x_2 \ \ldots\ \forall x_n\ A$, and the *existential closure* of A is $\exists x_1 \exists x_2 \ \ldots\ \exists x_n\ A$.

The notation $A(x_1, \ldots, x_n)$ indicates that some of the variables $x_1, \ldots, x_n$ are free variables in the formula A, but this will not mean that those are the only free variables in A. We use the notation so that we can write $A(t_1, \ldots, t_n)$ to denote the result of substituting the *terms* $t_1, \ldots, t_n$ for all free occurrences of the respective variables $x_1, \ldots, x_n$ in A. A term is either a variable or a constant symbol.
A term t is *free for* x_i *in* A if t is a constant symbol, or if t is a variable x_j, and no free occurrence of x_i in A lies within the scope of a quantifier $\forall x_j$ or $\exists x_j$ (meaning that x_j can be substituted for all free occurrences of x_j without creating any new bound occurrences of x_j).

Example 7.13 The formula $p(x, y)$ has two free variables, and $\exists y p(x, y)$ has one free variable, x. The formula $\forall x \exists y p(x, y)$ is closed. The universal closure of $p(x, y)$ is $\forall x \forall y\ p(x, y)$, and its existential closure is $\exists x \exists y\ p(x, y)$. □

Example 7.14 In $\forall x\ p(x) \wedge q(x)$, the first occurrence of x is bound and the second is free. The universal closure is $\forall x\ (\forall x\ p(x) \wedge q(x))$. However, it will be seen when the semantics of first-order formulas are defined in the next section that a bound occurrence of a variable is a "dummy" variable: the variable can be changed to another letter without affecting the interpretation of the formula. Thus, the above universal closure could equivalently be written as $\forall x\ (\forall y\ p(y) \wedge q(x))$. □

7.3 Interpretations

Definition 7.16 Let A be a formula containing the predicate symbols $p_1^{n_1}, \ldots, p_m^{n_m}$ and the constant symbols $a_1, \ldots, a_k$. An *interpretation* $\mathscr{I}_A$ for A is a triple

$$(D, \{R_1, \ldots, R_m\}, \{d_1, \ldots, d_k\}) ,$$

where D is a nonempty set called the *domain* of the interpretation, R_i is an n_i-ary relation on D that is assigned to the predicate symbol p_i, and an element $d_i \in D$ is assigned to the constant symbol a_i.

There exist logics with multiple domains – or 'types.' These are commonly called *multi-sorted* logics.

Example 7.17 Here are three different interpretations for the formula $\forall x\ p(a, x)$:

$$\mathscr{I}_1 = (\mathbb{N}, \{\leq\}, \{0\}), \qquad \mathscr{I}_2 = (\mathbb{N}, \{\leq\}, \{1\}), \qquad \mathscr{I}_3 = (\mathbb{Z}, \{\leq\}, \{0\}) .$$

Intuitively, under the first interpretation, the formula is true, while it is false under the other two. □

To allow for free variables we make the following

Definition 7.18 Let $\mathscr{I}_A$ be an interpretation for a formula A. An *assignment* $\sigma_{\mathscr{I}_A} : \mathscr{V} \longrightarrow D$ is a function that maps every free variable $v \in V$ to an element d of the domain D of the interpretation $\mathscr{I}_A$.

The map $\sigma_{\mathscr{I}_A}[x \leftarrow d]$ is an assignment that is the same as $\sigma_{\mathscr{I}_A}$, except that it maps x to d.

Here's an informal introduction to the semantics of first-order formulas. A formal definition will follow.

The interpretation $\mathscr{I}$ assigns concrete mathematical objects to the predicate symbols and the constant symbols. For formulas that don't contain variables or quantifiers, the interpretation therefore determines the truth values of atomic formulas. That in turn determines the truth value of the formula as a whole, through the truth tables for the Boolean operators.

For example, consider the formula $p(a, b) \wedge p(b, c) \Rightarrow p(a, c)$. If the interpretation $\mathscr{I}$ is $(\mathbb{N}, \{\leq\}, \{2, 3, 4\})$, then it interprets the formula over the domain of the natural numbers, making the binary predicate symbol denote the binary relation $\leq$, and making the constant symbols a, b, and c stand for the domain elements 2, 3, and 4, respectively. That means that the formula is interpreted to mean that $2 \leq 3$ and $3 \leq 4$ implies $2 \leq 4$, which happens to be true.

Now suppose that we wish to consider a formula that contains variables, but no quantifiers. Then the assignment $\sigma_{\mathscr{I}}$ plays a similar role to the interpretation; namely, it assigns a domain element to each variable. Together with the interpretation, that determines the truth values of atomic formulas, and therefore of the formula as a whole.

For example, let the formula A be $p(a, y) \wedge p(y, x) \Rightarrow p(a, x)$, and let the interpretation be the same as before. Then a still denotes 2, but what about x and y? That is where the assignment comes in. Consider the assignment $\sigma_{\mathscr{I}}$ that maps x to 4 and y to 3. Under $\mathscr{I}$ and $\sigma_{\mathscr{I}}$, the formula means the same thing as in the previous example.

But of course, we wish to define the semantics of quantified formulas, too. Suppose that a single variable, x, of a formula is universally quantified – say that the formula is $\forall x\ A$. Then, in order to determine an appropriate truth value for the formula, we need to consider all possible assignments of domain elements to the specific variable x, leaving the assignments to other variables unchanged. The formula $\forall x\ A$ should have the truth value T under an interpretation $\mathscr{I}$ and an assignment $\sigma_{\mathscr{I}}$, if and only if, whenever we reassign to x any domain element d, the formula A has the truth value T.

For instance, in the previous example, $\forall x\ A$ should have the truth value T under the above interpretation and assignment if and only if $2 \leq 3$ and $3 \leq d$ imply $2 \leq d$, for all $d \in D$ – in other words, for every natural number d. That implication is indeed true for any d, so the quantified formula should have the value T.

So we need to consider not just the specific value assigned to x by the assignment $\sigma_{\mathscr{I}}$, but all possible values that could be assigned to that variable, and determine whether all of those possible values lead to the satisfaction of the formula. That is why we bring in the operation $[x \leftarrow d]$ – it changes the value that $\sigma_{\mathscr{I}}$ assigns to the specific variable x, while leaving the values assigned to other variables fixed. For any other variable y, $\sigma_{\mathscr{I}}[x \leftarrow d](y) = \sigma_{\mathscr{I}}(y)$, but $\sigma_{\mathscr{I}}[x \leftarrow d](x) = d$.

The formula $\forall x\ A$ should therefore have the truth value T under the interpretation $\mathscr{I}$ and the assignment $\sigma_{\mathscr{I}}$ if and only if A has the truth value T under $\mathscr{I}$ and $\sigma_{\mathscr{I}}[x \leftarrow d]$, for all $d \in D$.

In other words, $\forall x\ A$ is assigned the truth value T under interpretation $\mathscr{I}$ and assignment $\sigma_{\mathscr{I}}$ if and only if, when the assignments to all other variables are held constant, but the value of x is allowed to range over the whole domain D of the interpretation, the truth value of A is always T.

We can now give a formal definition of the truth values of first-order formulas.

Definition 7.19 Let A be a formula, $\mathscr{I}_A$ an interpretation for A, and $\sigma_{\mathscr{I}_A}$ an assignment. Then $v_{\sigma_{\mathscr{I}_A}}(A)$, the *truth value* of A under $\mathscr{I}_A$ and $\sigma_{\mathscr{I}_A}$, is defined recursively as follows (with $v_{\sigma_{\mathscr{I}_A}}$ replaced by v_σ):

- Let $A = p_k(c_1, \ \ldots \ , c_n)$ be an atomic formula, where each c_i is either a constant symbol or a variable. Then $v_\sigma(A) = T$ iff $(d_1, \ \ldots \ , d_n) \in R_k$, where R_k is the relation assigned to p_k by $\mathscr{I}_A$, and d_i is the domain element assigned to c_i, by $\mathscr{I}_A$ if c_i is a constant symbol, or by $\sigma_{\mathscr{I}_A}$, if c_i is a variable.
- $v_\sigma(\neg A_1) = T$ iff $v_\sigma(A_1) = F$.
- $v_\sigma(A_1 \vee A_2) = T$ iff $v_\sigma(A_1) = T$ or $v_\sigma(A_2) = T$; and similarly for the other Boolean operators.
- $v_\sigma(\forall x A_1) = T$ iff $v_{\sigma[x \leftarrow d]}(A_1) = T$ for all $d \in D$.
- $v_\sigma(\exists x A_1) = T$ iff $v_{\sigma[x \leftarrow d]}(A_1) = T$ for some $d \in D$.

Theorem 7.20 Let A be a closed formula, $\mathscr{I}_A$ be an interpretation for A and $\sigma_{\mathscr{I}_A}$ an assignment. Then $v_{\sigma_{\mathscr{I}_A}}(A)$ does not depend on the assignment $\sigma_{\mathscr{I}_A}$.

By the Theorem, if A is a closed formula, we can denote its truth value $v_{\mathscr{I}_A}(A)$, without mentioning an assignment.

Example 7.21 The formal semantics conform to the intuitive conclusions reached in Example 7.17:

- $v_{\mathscr{I}_1}(A) = T$: zero is the smallest natural number;
- $v_{\mathscr{I}_2}(A) = F$: 1 is not the smallest natural number;
- $v_{\mathscr{I}_3}(A) = F$: there is no smallest integer.

□

Theorem 7.22 Let A be a formula with free variables $x_1, \ldots, x_n$, and let $\mathscr{I}_A$ be an interpretation for A. Then:

1. $v_{\sigma_{\mathscr{I}_A}}(A) = T$ for some assignment $\sigma_{\mathscr{I}_A}$ iff $v_{\mathscr{I}}(\exists x_1 \ \ldots \ \exists x_n A) = T$; and
2. $v_{\sigma_{\mathscr{I}_A}}(A) = T$ for all assignments $\sigma_{\mathscr{I}_A}$ iff $v_{\mathscr{I}}(\forall x_1 \ \ldots \ \forall x_n A) = T$.

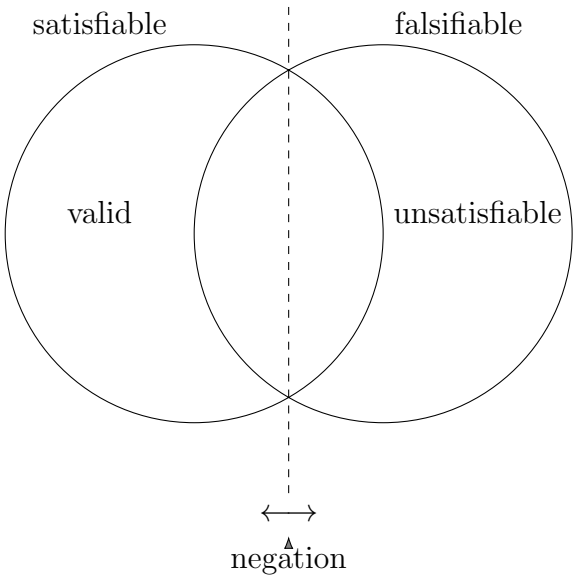
7.3.2 Validity and satisfiability

Definition 7.23 Let A be a formula of first-order logic:

- A is *true* in $\mathscr{I}$ or $\mathscr{I}$ is a *model* for A if $v_{\sigma_{\mathscr{I}}}(A) = T$ for all assignments. This may be written $\mathscr{I} \vDash A$.
- A is *valid* if for all interpretations $\mathscr{I}$, $\mathscr{I} \vDash A$. This may be written $\vDash A$.
- A is *satisfiable* if for some interpretation $\mathscr{I}$, $\mathscr{I} \vDash A$.
- A is *unsatisfiable* if it is not satisfiable.
- A is *falsifiable* if it is not valid.

Note: this definition differs from that in the text, mainly because it applies to all formulas, whether closed or not. Formulas with free variables are treated as if those variables are universally quantified.

The interrelationships among these concepts may be displayed as in propositional logic:



Example 7.24 The closed formula $\forall x\ p(x) \Rightarrow p(a)$ is valid.

Suppose not. Then there is an interpretation $\mathscr{I} = (D, \{R\}, \{d\})$ such that $v_{\mathscr{I}}(\forall x\ p(x)) = T$ and $v_{\mathscr{I}}(p(a)) = F$. But, by definition, the former implies that $v_{\sigma_{\mathscr{I}}[x \leftarrow d]}(p(x)) = T$, and that holds if and only if $d \in R$, which means $v_{\mathscr{I}}(p(a)) = T$, a contradiction. □

Example 7.25

- $\forall x \forall y\ (p(x, y) \Rightarrow p(y, x))$.
  This formula is true in any interpretation which assigns p a symmetric binary relation. But it is not valid because it is false whenever p is assigned a non-symmetric relation.

- $\forall x \exists y\ p(x, y)$.
  This formula is true, for example, in any interpretation in which p is assigned to a total function. But it is not valid: suppose that $\mathscr{I} = (\mathbb{N}, \{>\}, \emptyset\}$; there is no natural number less than zero.

- $\exists x \exists y\ (p(x) \wedge \neg p(y))$.
  This formula can only be true in interpretations over domains with at least two elements.

- $\forall x\ p(a, x)$.
  This formula is true only in an interpretation $\mathcal{I}$ that assigns a binary relation R to p and an element d of the domain to a such that d is related by R to every other element of the domain.

- $\forall x\ (p(x) \wedge q(x)) \iff (\forall x\ p(x)) \wedge \forall x\ q(x))$.
  This formula is valid. It expresses two different ways of asserting that p and q are assigned the whole domain D of the interpretation.

- $\forall x\ (p(x) \implies q(x)) \implies (\forall x\ p(x) \implies \forall x\ q(x))$.
  This formula is also valid, but its converse is not.

□

7.3.3 An interpretation for a set of formulas

This notion extends straightforwardly from propositional to predicate logic. An interpretation for a set of formulas is one that assigns relations over its domain to all predicate symbols occurring in the set of formulas, and assigns elements of the domain to all constant symbols occurring in the set of formulas. It satisfies the set of formulas, or is a model for the set of formulas, if it is a model for each formula in the set.

7.4 Logical equivalence and logical consequence

The formulas A_1 and A_2 are logically equivalent if any interpretation $\mathscr{I}$ for the set $\{A_1, A_2\}$ of the two formulas is a model for A_1 iff it is a model for A_2.

If A is a formula and U a set of formulas, then A is a logical consequence of U, written $U \vDash A$, if every interpretation $\mathscr{I}$ for $U \cup \{A\}$ that is a model for U is a model for A.

7.4.1 Logical equivalence in first-order logic

Exercise: One of the logical equivalences and valid implications given in this section of the textbook contains a typographical error. Find it. [The error does not make the result wrong, but it does make it weaker than necessary.]

8.2 Hilbert system $\mathscr{H}$

The axioms of $\mathscr{H}$ are:

Axiom 1 $\vdash (A \Rightarrow (B \Rightarrow A))$,

Axiom 2 $\vdash (A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$,

Axiom 3 $\vdash (\neg B \Rightarrow \neg A) \Rightarrow (A \Rightarrow B)$,

Axiom 4 $\vdash \forall x\ A(x) \Rightarrow A(t)$,

Axiom 5 $\vdash \forall x\ (A \Rightarrow B) \Rightarrow (A \Rightarrow \forall x\ B)$;

where,

- in Axioms 1, 2, and 3, A, B, and C are any formulas of first-order logic,
- in Axiom 4, t is a term that is free for x in A, and
- in Axiom 5, A is a formula containing no free occurrences of x.

The inference rules are *modus ponens* (MP) and *generalization* (Gen):

$$\frac{\vdash A \Rightarrow B \qquad \vdash A}{\vdash B}\ , \qquad\qquad \frac{\vdash A}{\vdash \forall x\ A}\ .$$

It can readily be checked that all of the axioms are valid formulas. Moreover, if the premises of an inference rule are valid, then so is its conclusion.

Note: Axiom 4, and the generalization inference rule, differ from those used in the text.
Note also that the atoms and inference rules do not explicitly refer to existential quantifiers. For the purpose of formal proofs, we shall not consider the existential quantifier to be a primitive symbol of the logical language; rather $\exists x\ A$ will be treated as an abbreviation of $\neg\forall x\ \neg A$. Such treatment is justified by the formal semantics.

Generalization and specialization

The new inference rule, generalization, formalizes a common inference: if A can be shown to hold *for an arbitrary value of* x, then it can be concluded that $\forall x\ A(x)$ holds.

Note that Axiom 4 immediately yields a derived rule that plays the role of the converse of the generalization inference rule:

Rule 8.6 (Specialization)

$$\frac{\vdash \forall x\ A(x)}{\vdash A(x)}\ .$$

The Deduction Rule

The statement of the Deduction Rule must be qualified so as to take account of the manner in which the generalization rule (Gen) may have been used in the deduction.

Because our semantics treats formulas with free variables as if those variables were universally quantified, the inference

$$\frac{\vdash A}{\vdash \forall x\ A}$$

is sound, but the implication $A \Rightarrow \forall x\ A$ is not generally valid.

In order to qualify the statement of the Deduction Rule appropriately, define a property of *dependence* of one formula on another in a deduction. In a proof $B_1 B_2 \ldots B_n$ of B from $U \cup \{A\}$, say that B_i *depends* on A if either

1. B_i is A, or
2. B_i is inferred via MP or Gen from earlier formulas in the sequence, at least one of which depends on A.

Rule 8.9 (Deduction Rule)
Suppose that there exists a deduction of B from $U \cup \{A\}$ in which no application of Gen to a formula that depends on A has as its quantified variable a variable that occurs free in A. Then $U \vdash A \Rightarrow B$.

Proof: By an induction on the length of the deduction. □

The qualification of the Deduction Rule reflects the fact that one can only generalize what has been established for arbitrary elements.

Simpler versions of the Deduction Rule apply under special circumstances:

Deduction Rule, 1st Corollary: Suppose that there exists a deduction of B from $U \cup \{A\}$ in which no application of Gen has as its quantified variable a variable that occurs free in A. Then $U \vdash A \Rightarrow B$. $\square$

Deduction Rule, 2nd Corollary: If A is a closed formula and $U \cup \{A\} \vdash B$, then $U \vdash A \Rightarrow B$. $\square$

Our treatment of semantics, the deductive system, and the deduction rule are based on Elliott Mendelson, *Introduction to Mathematical Logic*, Chapman and Hall, 4th edition.

9.1 First-order logic with functions

While the fragment of first-order logic presented so far is an important one, especially for database applications, we shall now bring in functions, for greater expressivity.

Definition 9.1 Let $\mathscr{F}$ be a countable set of *function symbols*, each with an arity of at least one. Then *terms* are defined as follows:

- a variable or a constant symbol is a term;
- if f^n is an n-ary function symbol, and $t_1,\ t_2,\ \ldots\ , t_n$ are terms, then $f^n(t_1,\ t_2,\ \ldots\ , t_n)$ is a term.

Moreover, the definition of an *atomic formula* is generalized to an n-ary predicate followed by a list of n arguments, each of which is a term in the above sense.

Example 9.2: Examples of terms are

$$a,\ x,\ f(a,x),\ f(g(x),y),\ g(f(a,g(b)))\ ;$$

and examples of atomic formulas are

$$p(a,b),\ p(x,f(a,x)),\ q(f(a,a),f(g(x),g(x)))\ .$$

□

The definitions of free and bound occurrences of variables, and of the free variables of a formula, remain unchanged, but a term t that is *free for* x_i in a formula A is now one that contains no variables x_j that fall under the scope of a quantifier $\forall x_j$ or $\exists x_j$ when t is substituted for all free occurrences of x_i in A.

The addition of function symbols to the language of first-order logic requires an extension of the definition of an interpretation:

Definition 9.3 Let A be a formula and suppose that the predicate symbols occurring in A are $p_1,\ \ldots\ , p_k$, the functions symbols in A are $f_1^{n_1},\ \ldots\ , f_l^{n_l}$ and the constant symbols occurring in A are $a_1,\ \ldots\ , a_m$. An *interpretation* $\mathscr{I}_A$ is a 4-tuple

$$(D, \{R_1,\ \ldots\ , R_k\}, \{F_1^{n_1},\ \ldots\ , F_l^{n_l}\},\ \{d_1,\ \ldots\ , d_m\})\ .$$

consisting of a nonempty *domain* D, an assignment of an n_i-ary relation over D to each predicate symbol p_i, an assignment of an n_j-ary function symbol $F_j^{n_j}$ to each function symbol f_j, and an assignment of an element $d_n \in D$ to each constant symbol a_n.

The rest of the definitions of semantics are unchanged, except that of the truth value of an atomic formula under an interpretation $\mathscr{I}_A$ and an assignment $\sigma_{\mathscr{I}_A}$.

For that, we extend the map $\sigma_{\mathscr{I}_A}$ to all terms, recursively:

- for any variable x, the definition of $\sigma_{\mathscr{I}_A}(x)$ is unchanged;
- for any constant symbol a_n in A, $\sigma_{\mathscr{I}_A}(a_n)$ is the domain element d_n assigned to a_n by the interpretation; and
- for any term $t = f_j^{n_j}(t_1, \ \ldots \ , t_{n_j})$ occurring in A,

$$\sigma_{\mathscr{I}_A}(t) = F_j^{n_j}(\sigma_{\mathscr{I}_A}(t_1)), \ \ldots \ , \sigma_{\mathscr{I}_A}(t_{n_j})) \ ,$$

  where $F_j^{n_j}$ is the function to which the n_j-ary function symbol f_j is assigned by the interpretation.

Now, the truth value assigned to an atomic formula $A = p_i(t_1, \ \ldots \ , t_{n_i})$ is defined so that

- $v_{\sigma_{\mathscr{I}_A}}(A) = T$ iff $(\sigma_{\mathscr{I}_A}(t_1), \ \ldots \ , \sigma_{\mathscr{I}_A}(t_{n_i})) \in R_i$,
  where R_i is the n_i-ary relation assigned to p_i by the interpretation $\mathscr{I}_A$.

The definitions of a formula that is *true* for an interpretation $\mathscr{I}$, of a *model* of a formula A, of *valid*, *satisfiable*, *unsatisfiable*, and *falsifiable* formulas are as before.

The definition of the Hilbert deductive system $\mathscr{H}$ can also be carried over unchanged to first-order logic with functions. So can the statement and the proof of the Deduction Rule.

8.4 Proofs of Theorems in $\mathscr{H}$

The proof system $\mathscr{H}$ for first-order logic subsumes that for propositional logic: all of the axioms, and the inference rule MP for propositional logic also belong to the deductive system for first-order logic; the theorems and derived rules of propositional logic therefore carry over to the first-order system.

Theorem 8.14

$$\vdash A(t) \Rightarrow \exists x\ A(x)\ ,$$

where t is any term that is free for x in $A(x)$.

Proof:

1. $\vdash \forall x\ \neg A(x) \Rightarrow \neg A(t)$	Axiom 4
2. $\vdash \neg\neg A(t) \Rightarrow \neg\forall x\ \neg A(x)$	Contrapositive Rule 1
3. $\vdash A(t) \Rightarrow \exists x\ A(x)$	Double Negation Rule 2

□

Theorem 8.15

$$\vdash \forall x\ A(x) \Rightarrow \exists x\ A(x)\ .$$

Proof: By definition, x is a term that is free for x in $A(x)$.

1. $\vdash \forall x\ A(x) \Rightarrow A(x)$	Axiom 4
2. $\vdash A(x) \Rightarrow \exists x\ A(x)$	Theorem 8.14
3. $\vdash \forall x\ A(x) \Rightarrow \exists x\ A(x)$	Transitivity Rule 1, 2

□

Theorem 8.16

$$\vdash \forall x\ (A(x) \Rightarrow B(x)) \Rightarrow (\forall x\ A(x) \Rightarrow \forall x\ B(x))\ .$$

Proof:

1. $\vdash \forall x\ A(x) \Rightarrow A(x)$	Axiom 4
2. $\{\forall x\ A(x)\} \vdash \forall x\ A(x)$	Assumption
3. $\{\forall x\ A(x)\} \vdash A(x)$	MP 1, 2
4. $\forall x\ (A(x) \Rightarrow B(x)) \Rightarrow (A(x) \Rightarrow B(x))$	Axiom 4
5. $\{\forall x\ (A(x) \Rightarrow B(x))\} \vdash \forall x\ (A(x) \Rightarrow B(x))$	Assumption
6. $\{\forall x\ (A(x) \Rightarrow B(x))\} \vdash A(x) \Rightarrow B(x)$	MP 4, 5
7. $\{\forall x\ (A(x) \Rightarrow B(x)), \forall x\ A(x)\} \vdash B(x)$	MP 3, 6
8. $\{\forall x\ (A(x) \Rightarrow B(x)), \forall x\ A(x)\} \vdash \forall x\ B(x)$	Gen 7
9. $\{\forall x\ (A(x) \Rightarrow B(x))\} \vdash \forall x\ A(x) \Rightarrow \forall x\ B(x)$	Deduction, 1st Cor. 8
10. $\vdash \forall x\ (A(x) \Rightarrow B(x)) \Rightarrow (\forall x\ A(x) \Rightarrow \forall x\ B(x))$	Deduction, 1st Cor. 9

□

[**Rule 8.17:** Our proof system is designed in such a way that Rule 8.17 of the textbook does not hold. Indeed, according to our definitions, the conclusion of the rule is not a logical consequence of the premise.]

Theorem 8.18

$$\vdash \exists x\ \forall y\ A(x,y) \implies \forall y\ \exists x\ A(x,y)\ .$$

Proof:

1.	$\vdash \forall y\ A(x,y) \Rightarrow A(x,y)$	Axiom 4
2.	$\vdash \neg A(x,y) \Rightarrow \neg\forall y\ A(x,y)$	Contrapositive 1
3.	$\vdash \forall x\ (\neg A(x,y) \Rightarrow \neg\forall y\ A(x,y))$	Gen 2
4.	$\vdash \forall x\ (\neg A(x,y) \Rightarrow \neg\forall y\ A(x,y))$ $\implies (\forall x\ \neg A(x,y) \Rightarrow \forall x\ \neg\forall y\ A(x,y))$	Theorem 8.16
5.	$\vdash \forall x\ \neg A(x,y) \Rightarrow \forall x\ \neg\forall y\ A(x,y)$	MP 3, 4
6.	$\vdash \exists x\ \forall y\ A(x,y) \Rightarrow \exists x\ A(x,y)$	Contrapositive 5
7.	$\vdash \forall y\ [\exists x\ \forall y\ A(x,y) \Rightarrow \exists x\ A(x,y)]$	Gen 6
8.	$\vdash \forall y\ [\exists x\ \forall y\ A(x,y) \Rightarrow \exists x\ A(x,y)]$ $\implies [\exists x\ \forall y\ A(x,y) \Rightarrow \forall y\ \exists x\ A(x,y)]$	Axiom 5
9.	$\vdash \exists x\ \forall y\ A(x,y) \Rightarrow \forall y\ \exists x\ A(x,y)$	MP 7, 8

□

Theorem 8.19 Let A be a formula containing no free occurrences of x.

$$\vdash \forall x\ (A \Rightarrow B(x)) \iff (A \Rightarrow \forall x\ B(x))\ ;$$
$$\vdash \exists x\ (A \Rightarrow B(x)) \iff (A \Rightarrow \exists x\ B(x))\ .$$

Proof: The proof of the first theorem is an exercise.

Proof sketch for the ($\Rightarrow$) conjunct of the second theorem:

1. $\{\neg(A \Rightarrow \exists x\ B(x)\} \vdash A \wedge \forall x\ \neg B(x)$	Abbreviation
2. $\{\neg(A \Rightarrow \exists x\ B(x)\} \vdash A \wedge \neg B(x)$	Specialization 1
3. $\{\neg(A \Rightarrow \exists x\ B(x)\} \vdash \forall x\ (A \wedge \neg B(x))$	Gen 2
4. $\{\neg(A \Rightarrow \exists x\ B(x)\} \vdash \forall x\ \neg(A \Rightarrow B(x))$	Abbreviation 3
5. $\{\neg(A \Rightarrow \exists x\ B(x)\} \vdash \neg\exists x\ (A \Rightarrow B(x))$	Abbreviation 4
6. $\vdash \neg(A \Rightarrow \exists x\ B(x)) \Rightarrow \neg\exists x\ (A \Rightarrow B(x))$	Deduction, 1st Cor. 5
7. $\vdash \exists x\ (A \Rightarrow B(x)) \Rightarrow (A \Rightarrow \exists x\ B(x))$	Contrapositive 6

Proof sketch for the reverse implication:

1. $\{\forall x\ \neg(A \Rightarrow B(x))\} \vdash \forall x\ \neg(A \Rightarrow B(x))$	Assumption
2. $\{\forall x\ \neg(A \Rightarrow B(x))\} \vdash \neg(A \Rightarrow B(x))$	Specialization 1
3. $\{\forall x\ \neg(A \Rightarrow B(x))\} \vdash A \wedge \neg B(x))$	Abbreviation 2
4. $\{\forall x\ \neg(A \Rightarrow B(x))\} \vdash A \wedge \forall x\ \neg B(x))$	Gen 3
5. $\vdash \forall x\ \neg(A \Rightarrow B(x)) \Rightarrow A \wedge \forall x\ \neg B(x)$	Deduction, 1st Cor. 4
6. $\vdash (A \Rightarrow \exists x\ B(x)) \Rightarrow \exists x\ (A \Rightarrow B(x))$	Contrapositive 5

□

Theorem 8.20

$$\vdash \forall x\ A(x) \iff \forall y\ A(y)\ .$$

Proof: See the textbook. □

Theorem 8.21 Let B be a formula that contains no free occurrences of x.

$$\vdash \forall x\ (A(x) \Rightarrow B) \iff (\exists x\ A(x) \Rightarrow B)$$

Proof: See the textbook. □

Strong Soundness and Strong Completeness

The deductive system $\mathscr{H}$ is designed in such a way that a formula can be deduced if and only if its universal closure can be deduced, so in defining soundness and completeness, it suffices to consider closed formulas. Given any set U of formulas, and a closed formula A, we shall again say that a deductive system satisfies *strong soundness* if

$$U \vdash A \text{ implies } U \models A$$

– in other words, if every formula A that can be deduced from U is a logical consequence of U. The deductive system satisfies *strong completeness* if

$$U \models A \text{ implies } U \vdash A$$

– in other words, if every logical consequence of U can be deduced from U.

The axioms of $\mathscr{H}$ are valid formulas. Moreover, the inference rules of $\mathscr{H}$ preserve logical consequence: if the premises of an inference rule are logical consequences of U, then so is its conclusion. Strong soundness of $\mathscr{H}$ follows.

A proof of strong completeness is similar to that for the propositional Hilbert system (see the solutions of assignment 2). As in the propositional case, a set U of formulas is *consistent* if there is no formula A such that $U \vdash A$ and $U \vdash \neg A$; and the proof of completeness involves showing that every consistent set has a model. Much as in the propositional case, we shall first show that any consistent set can be extended to one that is *negation complete*. But here, we need to define a domain for the semantic interpretation, with nothing to go on but the syntactic property that the set is consistent. Define a *ground term* to be a term that contains no variables. A suitable domain consists of all ground terms of the set of formulas, provided that the set of formulas *contains witnesses*.

We say that a set U of formulas *contains witnesses* if for every first-order formula $B(x)$ containing a single free variable x, there is a ground term t such that

$$U \vdash \exists x\ B(x) \Rightarrow B(t)\ .$$

The ground term is said to be a *witness* because in any model of the set, if the existentially quantified formula is true, then so is $B(t)$; the ground term t denotes a specific domain element that satisfies the existentially quantified formula. Indeed, in our case, the ground term will actually *be* such a domain element.

Henkin Witnesses Lemma Every consistent set U of formulas has a consistent superset U' (over an extended language that includes a countable set of new constant symbols) that contains witnesses and contains countably many ground terms.

Proof: Let $c_1,\ c_2,\ c_3,\ \ldots$ be a sequence of constant symbols that do not appear in U, and let $B_1(x_{i_1}), B_2(x_{i_2}), B_3(x_{i_3}),\ \ldots$ be an enumeration of all formulas with only a single free variable.

Define the nondecreasing sequence of sets of formulas

$$U_0 \subseteq U_1 \subseteq U_2 \subseteq\ \ldots$$

as follows:

$$\begin{aligned} &U_0 := U\ , \\ &U_j := U_{j-1} \cup \{\exists x\ B_j(x_{i_j}) \Rightarrow B_i(c_j)\}\ , \qquad\qquad \text{for all } j > 0. \end{aligned}$$

Let U' be the union of all the U_i. Then U' is consistent if and only if all of the U_j are (because a deduction based on U' can only employ a finite number of assumptions from that set). Suppose that the latter does not hold. Because U_0 is consistent by assumption, let j be the least natural number so that U_{j-1} is consistent but U_j is not. Then, because any formula can be deduced from an inconsistent set,

$$\begin{aligned} &U_j \vdash \neg(\exists x\ B_j(x_{i_j}) \Rightarrow B_j(c_j)) \\ &\text{iff } U_{j-1} \cup \{\exists x\ B_j(x_{i_j}) \Rightarrow B_j(c_j)\} \vdash \neg(\exists x\ B_j(x_{i_j}) \Rightarrow B_j(c_j)) \\ &\text{iff } U_{j-1} \vdash (\exists x\ B_j(x_{i_j}) \Rightarrow B_j(c_j)) \Rightarrow \neg(\exists x\ B_j(x_{i_j}) \Rightarrow B_j(c_j)) \quad \text{Ded'n, 2nd Cor.} \\ &\text{iff } U_{j-1} \vdash \neg(\exists x\ B_j(x_{i_j}) \Rightarrow B_j(c_j)) \\ &\text{so } U_{j-1} \vdash \exists x\ B_j(x_{i_j}) \ \&\ U_{j-1} \vdash \neg B_j(c_j) \end{aligned}$$

But, because c_j does not occur in U_{j-1}, it can only have been introduced by an invocation of an axiom in the deduction of $B_j(c_j)$. Replace it everywhere in that deduction with a variable y, that does not occur in the original deduction. Then, by Gen, $U_{j-1} \vdash \forall y\ \neg B_j(y)$ – that is, $U_{j-1} \vdash \neg\exists y\ B_j(y)$. By Theorem 8.20, this contradicts the consistency of U_{j-1}. Therefore, $U' \supseteq U$ is consistent. Moreover, by construction, it contains witnesses. □

A set U of formulas is *negation-complete* if, for every formula A, either $U \vdash A$ or $U \vdash \neg A$.

Lindenbaum's Lemma Every consistent set of formulas extends to a negation-complete set of formulas.

Proof: Similar to that of the corresponding result for the propositional case. (See the solution of assignment 2.) □

Model-Existence Lemma Let U be a negation-complete set of formulas that contains witnesses. Then U has a model whose domain is the set of ground terms of U.

Proof: Let each constant symbol be assigned to itself, and any function symbol $f_j^{n_j}$ to the function that maps any n_j-tuple of closed terms $(t_1, \ \ldots \ , t_{n_j})$ to the closed term $f_j^{n_j}(t_1, \ \ldots \ , t_{n_j})$. Any predicate symbol $p_i^{n_i}$ is assigned to the set of n_i-tuples of closed terms $(t_1, \ \ldots \ , t_{n_i})$ such that $U \vdash p_i^{n_i}(t_1, \ \ldots \ , t_{n_i})$. Call the resulting interpretation $\mathscr{I}$. It can be shown by structural induction that, for any closed formula A of U,

$$\mathscr{I} \vDash A \text{ iff } U \vdash A \ .$$

For details, see Elliott Mendelson, *Introduction to Mathematical Logic*, Chapman and Hall, 4th edition. □

Say that an interpretation is a *countable model* of a set of formulas if it is a model of the set and its domain is a countable set.

Theorem Every consistent set has a countable model.

Proof: Applying the Henkin Witnesses Lemma we get a consistent superset that contains witnesses and has a countable set of closed terms. Lindenbaum's Lemma can be applied to that set to get a superset that is negation-complete; that superset too will contain witnesses (as a superset of a set that already contained witnesses), and by the proof of Lindenbaum's Lemma, it too will contain a countable set of closed terms. The Model-Existence Lemma then implies the existence of a countable model. □

Strong Completeness Theorem: Let U be a set of formulas and A a closed formula. Then

$$U \vDash A \text{ implies } U \vdash A \ .$$

Proof: Suppose that

$$U \vDash A \ .$$

Now assume that A cannot be deduced from U. Then $U \cup \{\neg A\}$ is consistent; otherwise A can be deduced from it, and then, by the 2nd Corollary of the Deduction Rule, $U \vdash \neg A \Rightarrow A$, which contradicts our assumption. By the preceding theorem, $U \cup \{\neg A\}$ has a countable model. That contradicts $U \vDash A$, so we must in fact have $U \vdash A$. □

The completeness of $\mathscr{H}$ was first proved by Gödel; strong completeness was then proved by Henkin, using a proof like that outlined above.

Strong completeness is important, in particular for mathematical applications, where the set U might represent mathematical axioms, such as those of Boolean algebra, or number theory. Hilbert envisioned using such a set of axioms to automate the answering of mathematical questions. One roadblock to Hilbert's program is the undecidability discussed at the beginning of this chapter. We'll conclude with another impediment discovered by Gödel. Though Gödel proved the completeness of first-order logic, he also showed that mathematical logic is *incomplete* in a different sense (that of the failure of negation-completeness). His incompleteness results shed further light on the results we've just considered.

Gödel's 'incompleteness' theorem: a rough outline

It's just been shown that $\mathscr{H}$ satisfies strong soundness and strong completeness:

$$U \vdash A \text{ if and only if } U \vDash A \ ;$$

a formula A can be deduced from a set of assumptions U if and only if it is a logical consequence of U – every model of U is a model of A.

But Gödel showed that mathematical logic is nevertheless 'incomplete' in an important sense. His results apply to the case where U is a set of mathematical assumptions, or 'proper axioms' for which there exists an algorithm that lists all formulas that can be deduced from U (for the purposes of this section, we'll call those formulas the *theorems*). The existence of such an algorithm does not just mean that the set of all such formulas is countable, but that their enumeration can be carried out effectively; a general term for this property is *recursively enumerable*, and a theory whose theorems are recursively enumerable is said to be *effectively generated*.

If the set of theorems and its complement were *both* recursively enumerable, two algorithms could be run concurrently, one listing the theorems and the other the non-theorems. Any given formula would eventually appear in one list or the other. That means that membership in the set of theorems would be decidable; another way of saying the same thing is that the set of theorems would be not just recursively enumerable, but *recursive*. Equivalently, by strong soundness and strong completeness, the set of logical consequences of U would be decidable. This is not generally the case, as Turing showed using the undecidability of the halting problem, and Church showed independently by means of his *λ-calculus*.

Gödel devised an ingenious means of associating with every well-formed formula a unique natural number, now known as its "Gödel number." Consequently, any logic that supports sufficient number-theoretic concepts[1] also allows for reasoning about the formulas of the logic itself, and even about the deductive system. In particular, Gödel showed that such a logic allows one to write down a formula that essentially means, "this formula is not a theorem." Somewhat more precisely, if Gödel's formula is true – in the 'standard' interpretation of the formula over the natural numbers – then it is not a theorem; and, contrapositively, if it is a theorem, then it cannot be true in that standard interpretation. As long as the standard interpretation is a model for the U, Gödel's self-referential formula cannot be a theorem, by strong soundness. But then its negation cannot be a theorem either, for Gödel's formula is true in the standard interpretation, and its negation is false.

Of course, there is no contradiction between this 'incompleteness' result and strong completeness. If neither $U \vdash A$ nor $U \vdash \neg A$ holds, that is because neither $U \vDash A$ nor $U \vDash \neg A$ holds. If U represents a suitable set of assumptions or 'axioms' of number theory, then Gödel showed that there are formulas A that hold in some models of U but not in others. The 'standard' model is not the only model.

Neither do these results contradict Lindenbaum's Lemma, which shows that U can be extended to a consistent set that is negation-complete. Rather, they show that, under any such extension, the set of theorems that can be deduced must cease to be recursively enumerable.

If either A or its negation were a theorem, for all formulas A, and if the

[1]Specifically, it suffices to assume the axioms of Robinson arithmetic, a fragment of Peano arithmetic.

set of theorems was recursively enumerable, then that set would in fact be decidable: if an algorithm listed the theorems, then eventually either A or its negation would appear, determining whether or not A was provable. Five years before Turing's and Church's undecidability results, Gödel showed in 1931 that that is not generally the case in mathematical logic.

Gödel's, Church's and Turing's results are milestones of intellectual history that revolutionized our understanding of the capacities of 'formal reasoning.' But despite the fundamental limitations that they identified, such formal reasoning still forms the theoretical foundation of mathematics; and it continues to grow in importance as a means of design and analysis of computing systems.