14

❖

Predicate Logic

We now turn our attention to a generalization of propositional logic, called "predicate," or "first-order," logic. Predicates are functions of zero or more variables that return Boolean values. Thus predicates can be true sometimes and false sometimes, depending on the values of their arguments. For example, we shall find in predicate logic atomic operands such as $csg(C, S, G)$. Here, csg is the predicate name, and C, S, and G are arguments. We can think of this expression as a representation in logic of the database relation Course-Student-Grade of Fig. 8.1. It returns the value TRUE whenever the values of C, S, and G are such that student S got grade G in course C, and it returns FALSE otherwise.

Using predicates as atomic operands, instead of propositional variables, gives us a more powerful language than expressions involving only propositions. In fact, predicate logic is expressive enough to form the basis of a number of useful programming languages, such as Prolog (which stands for "*Pro*gramming in *log*ic") and the language SQL that we mentioned in Section 8.7. Predicate logic is also used in reasoning systems or "expert" systems, such as automatic medical diagnosis programs and theorem-proving programs.

❖ 14.1 What This Chapter Is About

We introduce predicates in Section 14.2. As we shall see, predicates provide much greater power to express ideas formally than do propositional variables. Much of the development of predicate logic parallels that of propositional logic in Chapter 12, although there are important differences.

✦   Expressions of predicate logic can be built from predicates using the operators of propositional logic (Section 14.3).

✦   "Quantifiers" are operators of predicate logic that have no counterpart in propositional logic (Section 14.4). We can use quantifiers to state that an expression is true for all values of some argument or that there exists at least one value of the argument that makes the expression true.

❖   "Interpretations" for expressions of predicate logic are possible meanings for the predicates and variables (Section 14.5). They are analogous to truth assignments in propositional logic.

❖   Tautologies of predicate logic are expressions that are true for all interpretations. Some tautologies of predicate logic are analogs of tautologies for propositional logic (Section 14.6), while others are not (Section 14.7).

❖   Proofs in predicate logic can be carried out in a manner similar to proofs in propositional logic (Sections 14.8 and 14.9).

In Section 14.10 we discuss some of the implications of predicate logic as to our ability to compute answers to questions. We shall discover the following:

❖   A statement's being a tautology does not mean that it is provable in certain proof systems.

❖   In particular, Gödel's incompleteness theorem tells us that there is a specialized form of predicate logic, dealing with the integers, in which no proof system can provide proofs of every tautology.

❖   Further, Turing's theorem tells us that there are problems we can state but cannot solve by any computer. An example is whether or not a given C program goes into an infinite loop on certain inputs.

❖❖ 14.2   Predicates

A predicate is a generalization of a propositional variable. Recalling Section 12.10, suppose that we have three propositions: r ("It is raining"), u ("Joe takes his umbrella"), and w ("Joe gets wet"). Suppose further that we have three hypotheses, or expressions that we assume are true: $r \rightarrow u$ ("If it rains, then Joe takes his umbrella"), $u \rightarrow \bar{w}$ ("If Joe takes an umbrella, then he doesn't get wet"), and $\bar{r} \rightarrow \bar{w}$ ("If it doesn't rain, Joe doesn't get wet").

What is true for Joe is also true for Mary, and Sue, and Bill, and so on. Thus we might think of the proposition u as u_{Joe}, while w is the proposition w_{Joe}. If we do, we have the hypotheses

$$r \rightarrow u_{Joe}, \ u_{Joe} \rightarrow \bar{w}_{Joe}, \text{ and } \bar{r} \rightarrow \bar{w}_{Joe}$$

If we define the proposition u_{Mary} to mean that Mary takes her umbrella, and w_{Mary} to mean that Mary gets wet, then we have the similar set of hypotheses

$$r \rightarrow u_{Mary}, \ u_{Mary} \rightarrow \bar{w}_{Mary}, \text{ and } \bar{r} \rightarrow \bar{w}_{Mary}$$

We could go on like this, inventing propositions to talk about every individual X we know of and stating the hypotheses that relate the proposition r to the new propositions u_X and w_X, namely,

$$r \rightarrow u_X, \ u_X \rightarrow \bar{w}_X, \text{ and } \bar{r} \rightarrow \bar{w}_X$$

We have now arrived at the notion of a predicate. Instead of an infinite collection of propositions u_X and w_X, we can define symbol u to be a predicate that takes an argument X. The expression $u(X)$ can be interpreted as saying "X takes his or her umbrella." Possibly, for some values of X, $u(X)$ is true, and for other values of X, $u(X)$ is false. Similarly, w can be a predicate; informally $w(X)$ says "X gets wet."

The propositional variable r can also be treated as a predicate with zero arguments. That is, whether it is raining does not depend on the individual X the way u and w do.

We can now write our hypotheses in terms of the predicates as follows:

1. $r \rightarrow u(X)$. (For any individual X, if it is raining, then X takes his or her umbrella.)

2. $u(X) \rightarrow \texttt{NOT}\ w(X)$. (No matter who you are, if you take your umbrella, then you won't get wet.)

3. $\texttt{NOT}\ r \rightarrow \texttt{NOT}\ w(X)$. (If it doesn't rain, then nobody gets wet.)

Atomic Formulas

An *atomic formula* is a predicate with zero or more arguments. For example, $u(X)$ is an atomic formula with predicate u and one argument, here occupied by the variable X. In general, an argument is either a variable or a constant.[1] While, in principle, we must allow any sort of value for a constant, we shall usually imagine that values are integers, reals, or character strings.

Variables and constants

Variables are symbols capable of taking on any constant as value. We should not confuse "variables" in this sense with "propositional variables," as used in Chapter 12. In fact, a propositional variable is equivalent to a predicate with no arguments, and we shall write p for an atomic formula with predicate name p and zero arguments.

Ground atomic formula

An atomic formula all of whose arguments are constants is called a *ground* atomic formula. Nonground atomic formulas can have constants or variables as arguments, but at least one argument must be a variable. Note that any proposition, being an atomic formula with no arguments, has "all arguments constant," and is therefore a ground atomic formula.

Distinguishing Constants From Variables

We shall use the following convention to distinguish constants from variables. A variable name will always begin with an upper-case letter. Constants are represented either by

1. Character strings beginning with a lower-case letter,

2. Numbers, like 12 or 14.3, or

3. Quoted character strings.

---

[1] Predicate logic also allows arguments that are more complicated expressions than single variables or constants. These are important for certain purposes that we do not discuss in this book. Therefore, in this chapter we shall only see variables and constants as arguments of predicates.

Thus, if we want to represent course CS101 by a constant, we could write it as "CS101", for example.[2]

Predicates, like constants, will be represented by character strings beginning with a lower-case letter. There is no possibility that we can confuse a predicate with a constant, since constants can only appear within argument lists in an atomic formula, while predicates cannot appear there.

✦ **Example 14.1.** We might invent a predicate name csg to represent the information contained in the Course-Student-Grade relation discussed in Section 8.2. The atomic formula $csg(C, S, G)$ can be thought of as saying, of variables C, S, and G, that student S took course C and got grade G. Put another way, when we substitute constants c for C, s for S, and g for G, the value of $csg(c, s, g)$ is TRUE if and only if student s took course c and got grade g.

We can also express the particular facts (i.e., tuples) in the relation as ground atomic formulas, by using constants as arguments. For instance, the first tuple of Fig. 8.1 could be expressed as $csg(\text{"CS101"}, 12345, \text{"A"})$, asserting that the student with ID 12345 got an A in CS101. Finally, we can mix constants and variables as arguments, so that we might see an atomic formula like $csg(\text{"CS101"}, S, G)$. This atomic formula is true if variables S and G take on any pair of values (s, g) such that s is a student who took course CS101, and got grade g and false otherwise. ✦

EXERCISES

14.2.1: Identify the following as constants, variables, ground atomic formulas, or nonground atomic formulas, using the conventions of this section.

a) CS205
b) cs205
c) 205
d) "cs205"
e) $p(X, x)$
f) $p(3, 4, 5)$
g) "$p(3, 4, 5)$"

❖❖ 14.3   Logical Expressions

The notions that we used in Chapter 12 for propositional logic — literals, logical expressions, clauses, and so on — carry over to predicate logic. In the next section we introduce two additional operators to form logical expressions. However, the basic idea behind the construction of logical expressions remains essentially the same in both propositional and predicate logic.

Literals

Ground literal

A *literal* is either an atomic formula or its negation. If there are no variables among the arguments of the atomic formula, then the literal is a *ground* literal.

---

[2] Constants are often called "atoms" in logic. Unfortunately, what we have referred to as "atomic formulas" are also called "atoms" at times. We shall generally avoid the term "atom."

◆ **Example 14.2.** $p(X, a)$ is an atomic formula and a literal. It is not ground because of the argument X, which is a variable by our convention. NOT $p(X, a)$ is a literal, but not an atomic formula, and not a ground literal. The expressions $p(a, b)$ and NOT $p(a, b)$ are ground literals; only the first is a (ground) atomic formula. ◆

As for propositional logic, we can use an overbar in place of the NOT operator. However, the bars become confusing to read when applied to a long expression, and we shall see NOT used more frequently in this chapter than in Chapter 12.

Logical Expressions

We can build expressions from atomic formulas just as we built expressions in Section 12.3 from propositional variables. We shall continue to use the operators AND, OR, NOT, $\rightarrow$, and $\equiv$, as well as other logical connectives discussed in Chapter 12. In the next section, we introduce "quantifiers," operators that can be used to construct expressions in predicate logic but have no counterpart in propositional logic.

As with the bar shorthand for NOT, we can continue to use the shorthands of juxtaposition (no operator) for AND and + for OR. However, we use these shorthands infrequently because they tend to make the longer expressions of predicate logic hard to understand.

The following example should give the reader some insight into the meaning of logical expressions. However, note that this discussion is a considerable oversimplification, and we shall have to wait until Section 14.5 to discuss "interpretations" and the meaning that they impart to logical expressions in predicate logic.

◆ **Example 14.3.** Suppose that we have predicates *csg* and *snap*, which we interpret as the relations Course-Student-Grade and Student-Name-Address-Phone that were introduced in Chapter 8. Suppose also that we want to find the grade of the student named "C. Brown" in course CS101. We could assert the logical expression

$$\big(csg(\text{"CS101"}, S, G) \text{ AND } snap(S, \text{"C. Brown"}, A, P)\big) \rightarrow answer(G) \quad (14.1)$$

Here, *answer* is another predicate, intended to be true of a grade G if G is the grade of some student named "C. Brown" in CS101.

When we "assert" an expression, we mean that its value is TRUE no matter what values we substitute for its variables. Informally, an expression such as (14.1) can be interpreted as follows. If we substitute a constant for each of the variables, then each of the atomic formulas becomes a ground atomic formula. We can decide whether a ground atomic formula is true or false by referring either to the "real world," or by looking it up in a relation that lists the true ground atomic formulas with a given predicate. When we substitute 0 or 1 for each of the ground atomic formulas, we can evaluate the expression itself, just as we did for propositional logic expressions in Chapter 12.

In the case of expression (14.1), we can take the tuples in Fig. 8.1 and 8.2(a) to be true. In particular,

$$csg(\text{"CS101"}, 12345, \text{"A"})$$

and

$$snap(12345, \text{``C. Brown''}, \text{``12 Apple St.''}, \text{``555-1234''})$$

are true. Then we can let

$S = 12345$
$G = \text{``A''}$
$A = \text{``12 Apple St.''}$
$P = \text{``555-1234''}$

That makes the left side of (14.1) become 1 AND 1, which has the value 1, of course. In principle, we don't know anything about the predicate *answer*. However, we asserted (14.1), which means that whatever values we substitute for its variables, its value is TRUE. Since its left side is made TRUE by the above substitution, the right side cannot be FALSE. Thus we deduce that *answer*("A") is true. ❖

Other Terminology

Clause

We shall use other terms associated with propositional logic as well. In general, when in Chapter 12 we spoke of propositional variables, in this chapter we speak of any atomic formula, including a predicate with zero arguments (i.e., a propositional variable) as a special case. For example, a *clause* is a collection of literals, connected by OR's. Similarly, an expression is said to be in *product-of-sums form* if it is the AND of clauses. We may also speak of *sum-of-products form,* where the expression is the OR of terms and each such term is the AND of literals.

EXERCISES

14.3.1: Write an expression similar to (14.1) for the question "What grade did L. Van Pelt get in PH100?" For what value of its argument is *answer* definitely true, assuming the facts of Figs. 8.1 and 8.2? What substitution for variables did you make to demonstrate the truth of this answer?

14.3.2: Let *cdh* be a predicate that stands for the Course-Day-Hour relation of Fig. 8.2(c), and *cr* a predicate for the Course-Room relation of Fig. 8.2(d). Write an expression similar to (14.1) for the question "Where is C. Brown 9AM Monday morning?" (More precisely, in what room does the course C. Brown is taking on Monday at 9AM meet?) For what value of its argument is *answer* definitely true, assuming the facts of Figs. 8.1 and 8.2? What substitution for variables did you make to demonstrate the truth of this answer?

14.3.3**: Each of the operations of relational algebra discussed in Section 8.7 can be expressed in predicate logic, using an expression like (14.1). For example, (14.1) itself is the equivalent of the relational algebra expression

$$\pi_{\text{Grade}}\big(\sigma_{Course=\text{``CS101''} \text{ AND } Name=\text{``C.Brown''}}(CSG \bowtie SNAP)\big)$$

Show how the effect of each of the operations selection, projection, join, union, intersection, and difference can be expressed in predicate logic in the form "expression implies answer." Then translate each of the relational algebra expressions found in the examples of Section 8.7 into logic.

❖❖ 14.4   Quantifiers

Let us return to our example involving the zero-argument predicate r ("It is raining") and the one-argument predicates $u(X)$ ("X takes his umbrella) and $w(X)$ ("X gets wet"). We might wish to assert that "If it rains, then somebody gets wet." Perhaps we could try

$$r \to w(\text{"Joe"}) \text{ OR } w(\text{"Sally"}) \text{ OR } w(\text{"Sue"}) \text{ OR } w(\text{"Sam"}) \text{ OR } \cdots$$

But this attempt fails because

1.   We can write as an expression the OR of any finite set of expressions, but we cannot write the OR of an infinite set of expressions.

2.   We don't know the complete set of individuals about whom we are speaking.

"There exists"   To express the OR of a collection of expressions formed by substituting every possible value for some variable X, we need an additional way to create expressions of predicate logic. The operator is $\exists$, read "there exists." We use it in expressions such as $(\exists X)w(X)$, or informally, "There exists an individual X such that X gets wet." In general, if E is any logical expression, then $(\exists X)(E)$ is also a logical expression.[3] Its informal meaning is that there is at least one value of X that makes E true. More precisely, for every value of E's other variables, we can find some value of X (not necessarily the same value in all cases) to make E true.

Similarly, we cannot write the infinite AND of expressions like

$$u(\text{"Joe"}) \text{ AND } u(\text{"Sally"}) \text{ AND } u(\text{"Sue"}) \text{ AND } u(\text{"Sam"}) \text{ AND } \cdots$$

"For all"   We instead need a symbol $\forall$ (read "for all") to let us construct the AND of the collection of expressions formed from a given expression by substituting all possible values for a given variable. We write $(\forall X)u(X)$ in this example to mean "for all X, X takes his or her umbrella." In general, for any logical expression E, the expression $(\forall X)(E)$ means that for all possible values of the other variables of E, every constant we may substitute for X makes E true.

Universal and existential quantifiers   The symbols $\forall$ and $\exists$ are called *quantifiers*. We sometimes call $\forall$ the *universal quantifier* and $\exists$ the *existential quantifier*.

❖   **Example 14.4.** The expression $r \to (\forall X)\big(u(X) \text{ OR } w(X)\big)$ means "If it rains, then for all individuals X, either X takes an umbrella or X gets wet." Note that quantifiers can apply to arbitrary expressions, not just to atomic formulas as was the case in previous examples.

For another example, we can interpret the expression

$$(\forall C)\Big(\big((\exists S)csg(C, S, \text{"A"})\big) \to \big((\exists T)csg(C, T, \text{"B"})\big)\Big) \tag{14.2}$$

---

[3]   The parentheses around the E are sometimes needed and sometimes not, depending on the expression. The matter will be clear when we discuss precedence and associativity of operators later in the section. The parentheses around $\exists X$ are part of the notation and are invariably required.

as saying, "For all courses C, if there exists a student S who gets an A in the course, then there must exist a student T who gets a B." Less formally, "If you give A's, then you also have to give B's."

A third example expression is

$$\big((\forall X)\ \text{NOT}\ w(X)\big)\ \text{OR}\ \big((\exists Y)w(Y)\big) \tag{14.3}$$

Informally, "Either all individuals X stay dry or, at least one individual Y gets wet." Expression (14.3) is different from the other two in this example, in that here we have a tautology — that is, an expression which is true, regardless of the meaning of predicate w. The truth of (14.3) has nothing to do with properties of "wetness." No matter what the set S of values that make predicate w true is, either S is empty (i.e., for all X, $w(X)$ is false) or S is not empty (i.e., there exists a Y for which $w(Y)$ is true). ◆

Recursive Definition of Logical Expressions

As a review, we shall give a recursive definition of the class of logical expressions in predicate logic.

BASIS. Every atomic formula is an expression.

INDUCTION. If E and F are logical expressions, then so are

1. NOT E, E AND F, E OR F, $E \rightarrow F$, and $E \equiv F$. Informally, we may allow other operators of propositional logic, such as NAND, to be used as well.

2. $(\exists X)E$ and $(\forall X)E$, for any variable X. In principle, X need not even appear in E, although in practice such expressions rarely "make sense."

Precedence of Operators

In general, we need to put parentheses around all uses of expressions E and F. However, as with the other algebras we have encountered, it is often possible to remove parentheses because of the precedence of operators. We continue to use the precedence of operators defined in Section 12.4, NOT (highest), AND, OR, $\rightarrow$, and $\equiv$ (lowest). However, quantifiers have highest precedence of all.

◆ **Example 14.5.** $(\exists X)p(X)$ OR $q(X)$ would be grouped

$$\big((\exists X)p(X)\big)\ \text{OR}\ q(X)$$

Similarly, the outer pairs of parentheses in (14.3) are redundant, and we could have written

$$(\forall X)\ \text{NOT}\ w(X)\ \text{OR}\ (\exists Y)w(Y)$$

We can also eliminate two pairs of parentheses from (14.2) and write it

$$(\forall C)\big((\exists S)csg(C, S, \text{"A"}) \rightarrow (\exists T)csg(C, T, \text{"B"})\big)$$

The pair of parentheses around the entire expression after the $(\forall C)$ is necessary so the "for all C" will apply to the entire expression. ◆

Order of Quantifiers

A common logical mistake is to confuse the order of quantifiers — for example, to think that $(\forall X)(\exists Y)$ means the same as $(\exists Y)(\forall X)$, which it does not. For example, if we informally interpret $loves(X, Y)$ as "X loves Y," then $(\forall X)(\exists Y)loves(X, Y)$ means "Everybody loves somebody," that is, for every individual X there is at least one individual Y that X loves. On the other hand, $(\exists Y)(\forall X)loves(X, Y)$ means that there is some individual Y who is loved by everyone — a very fortunate Y, if such a person exists.

Note that the parentheses around the quantifiers $(\forall X)$ and $(\exists X)$ are not used for grouping, and should be regarded as part of the symbol indicating a quantifier. Also, remember that the quantifiers and NOT are unary, prefix operators, and the only sensible way to group them is from the right.

◆ **Example 14.6.** Thus the expression $(\forall X)$ NOT $(\exists Y)p(X, Y)$ is grouped

$$(\forall X)\Big(\text{NOT}\ \big((\exists Y)p(X, Y)\big)\Big)$$

and means "For all X there is no Y such that $p(X, Y)$ is true." Put another way, there is no pair of values for X and Y that makes $p(X, Y)$ true. ◆

Bound and Free Variables

Quantifiers interact with the variables that appear in an expression in a subtle way. To address this issue, let us first recall the notion of local and global variables in C. Suppose X is defined as an external variable in a C program, as suggested by **Local and global variables in C** Fig. 14.1. Assuming X is not declared in `main`, the reference to X in `main` is to the external variable. On the other hand, X is declared in function `f` as a local (automatic) variable, and all references to X within `f` are to this local variable.

```
int X;
    ...
main()
{
    ...
    ++X;
    ...
}

void f()
{
    int X;
    ...
}
```
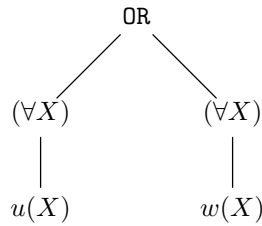
Fig. 14.1. Local and global variables.

Fig. 14.2. Expression tree for $(\forall X)u(X)$ OR $(\forall X)w(X)$.

There is a close analogy between a declaration of X in a C program on one hand, and a quantifier $(\forall X)$ or $(\exists X)$ on the other. If we have an expression $(\forall X)E$ or $(\exists X)E$, then the quantifier serves to declare X locally for the expression E, as if E were a function and X were declared local to that function.

In what follows, it helps to use the symbol Q to stand for either quantifier. Specifically, we take (QX) to stand for "some quantifier applied to X," that is, either $(\forall X)$ or $(\exists X)$.

If E has a subexpression of the form $(QX)F$, then this subexpression is like a block declared within E that has its own declaration of X. References to X within F refer to the X "declared" by this (QX), while uses of X within E but outside F refer to some other declaration of X — a quantifier either associated with E or with some expression contained within E but enclosing the use of X in question.

✦ **Example 14.7.** Consider the expression

$$(\forall X)u(X) \text{ OR } (\forall X)w(X) \tag{14.4}$$

Informally, "Either everyone takes an umbrella, or everyone gets wet." We might not believe in the truth of this statement, but let us consider it as an example. The expression tree for expression (14.4) is shown in Fig. 14.2. Note that the first quantifier $(\forall X)$ has only the use of X within u as its descendant, while the second $(\forall X)$ has only the use of X within w as a descendant. To tell at which quantifier a use of X is "declared," we have only to trace upwards from the use until we meet a quantifier (QX). Thus the two uses of X refer to different "declarations," and there is no relationship between them. ✦

Note we could have used different variables for the two "declarations" of X in (14.4), perhaps writing $(\forall X)u(X)$ OR $(\forall Y)w(Y)$. In general, we can always rename variables of a predicate logic expression so no one variable appears in two quantifiers. The situation is analogous to a programming language such as C, in which we can rename variables of a program, so that the same name is not used in two declarations. For example, in Fig. 14.1 we could change all instances of the variable name X in the function f to any new variable name Y.

✦ **Example 14.8.** For another example, consider the expression

$$(\forall X)\big(u(X) \text{ OR } (\exists X)w(X)\big)$$

Informally, "For each individual, either that individual takes his umbrella, or there exists some (perhaps other) individual who gets wet." The tree for this expression is shown in Fig. 14.3. Notice that the use of X within w refers to the closest enclosing "declaration" of X, which is the existential quantifier. Put another way, if we travel up the tree from $w(X)$, we meet the existential quantifier before we meet the universal quantifier. However, the use of X within u is not in the "scope" of the existential quantifier. If we proceed upward from $u(X)$, we first meet the universal quantifier. We could rewrite the expression as

$$(\forall X)\big(u(X) \text{ OR } (\exists Y)w(Y)\big)$$

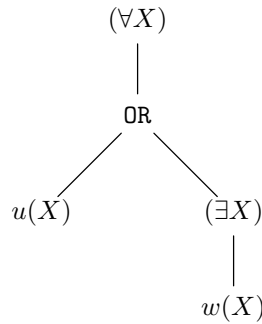so no variable is quantified more than once. ❖

Fig. 14.3. Expression tree for $(\forall X)\big(u(X) \text{ OR } (\exists X)w(X)\big)$.

Bound and free occurrences of variables

We say an occurrence of a variable X within a logical expression E is *bound* by a quantifier (QX) if, in the expression tree for E, that quantifier is the lowest ancestor of this occurrence of X that is a quantifier involving X. If an occurrence of X is not bound by any quantifier, then that occurrence of X is said to be *free*. Thus quantifiers act like "declarations" that are local to the subtree T rooted at the node of the quantifier. They apply everywhere within T, except within a subtree rooted at another quantifier with the same variable. Free variables are like variables global to a function, in that their "declaration," if there is one, occurs somewhere outside the expression in question.

❖ **Example 14.9.** Consider the expression

$$u(X) \text{ OR } (\exists X)w(X)$$

that is, "Either X takes his or her umbrella, or there is some person who gets wet." The tree is shown in Fig. 14.4. As in the previous examples, the two occurrences of X refer to different individuals. The occurrence of X in w is bound to the existential quantifier. However, there is no quantifier for X above the occurrence of X in u, and so this occurrence of X is free in the given expression. This example points up the fact that there can be both free and bound occurrences of the same variable, so that we must talk of "bound occurrences" rather than "bound variables," in some situations. The expressions in Examples 14.7 and 14.8 illustrate that it is also

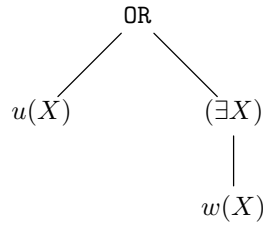possible for different occurrences of a variable to be bound to different occurrences of quantifiers. ◆

Fig. 14.4. Expression tree for $u(X)$ OR $(\exists X)w(X)$.

EXERCISES

14.4.1: Remove redundant pairs of parentheses from the following expressions.

a) $(\forall X)\left(\left(\exists Y\right)\left(\text{NOT}\left(p(X)\text{ OR }\left(p(Y)\text{ AND }q(X)\right)\right)\right)\right)$

b) $(\exists X)\left(\left(\text{NOT }p(X)\right)\text{ AND }\left((\exists Y)(p(Y))\text{ OR }\left(\exists X\right)\left(q(X,Z)\right)\right)\right)$

14.4.2: Draw expression trees for the expressions of Exercise 14.4.1. Indicate for each occurrence of a variable to which quantifier, if any, it is bound.

14.4.3: Rewrite the expression of Exercise 14.4.1(b) so that it does not quantify the same variable twice.

14.4.4*: In the box on "Order of Quantifiers," we spoke of a predicate $loves(X,Y)$, and gave it the expected informal interpretation. However, as we shall see in Section 14.5, predicates have no specific interpretation, and we could just as well have taken *loves* to talk about integers rather than individuals, and for $loves(X,Y)$ to have the informal interpretation that $Y = X + 1$. Under that interpretation, compare the meanings of the expressions $(\forall X)(\exists Y)loves(X,Y)$ and $(\exists Y)(\forall X)loves(X,Y)$. What are their informal interpretations? Which, if either, do you believe?

**14.4.5***: Using the *csg* predicate of our running example, write expressions that assert the following.

a)   C. Brown is an A student (i.e., he gets A's in all his courses).
b)   C. Brown is not an A student.

14.4.6*: Design a grammar that describes the legal expressions of predicate logic. You may use symbolic terminals like *constant* and *variable*, and you need not avoid redundant parentheses.

❖❖ 14.5   Interpretations

Until now, we have been rather vague about what an expression of predicate logic "means," or how we ascribe a meaning to an expression. We shall approach the subject by first recalling the "meaning" of a propositional logic expression E. That meaning is a function that takes a "truth assignment" (assignment of truth values 0 and 1 to the propositional variables in E) as its argument and produces 0 or 1 as its result. The result is determined by evaluating E with the atomic operands replaced by 0 or 1, according to the given truth assignment. Put another way, the meaning of a logical expression E is a truth table, which gives the value of E (0 or 1) for each truth assignment.

A truth assignment, in turn, is a function that takes propositional variables as arguments and returns 0 or 1 for each. Alternatively, we can see a truth assignment as a table that gives, for each propositional variable, a truth value, 0 or 1. Figure 14.5 suggests the role of these two kinds of functions.

$$p \quad \xrightarrow[\text{assignment}]{\text{truth}} \quad 0 \text{ or } 1$$
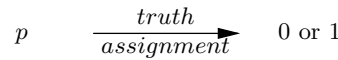
(a) A truth assignment is a function from predicates to truth values.

$$\begin{array}{c}\text{truth} \\ \text{assignment}\end{array} \quad \xrightarrow{\text{meaning}} \quad 0 \text{ or } 1$$
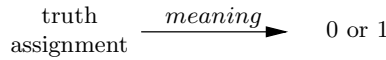
(b) The meaning of an expression is a function from truth assignments to truth values.

Fig. 14.5. The meaning of expressions in propositional logic.

In predicate logic, it is not sufficient to assign a constant 0 or 1 (TRUE or FALSE) to predicates (unless they have no arguments, in which case they are essentially propositional variables). Rather, the value assigned to a predicate is itself a function that takes values of the predicate's arguments as its own input, and produces 0 or 1 as output.

Domain   More precisely, we must first pick a nonempty *domain D* of values, from which we can select values for the variables. This domain could be anything: integers, reals, or some set of values with no particular name or significance. We assume, however, that the domain includes any constants appearing in the expression itself.

Now, let p be a predicate with k arguments. Then an *interpretation for predicate p* is a function that takes as input an assignment of domain elements to each of the k arguments of p and returns 0 or 1 (TRUE or FALSE). Equivalently, we can see the interpretation of p as a relation with k columns. For each assignment of values to the arguments that makes p true in this interpretation, there is a tuple of the relation.[4]
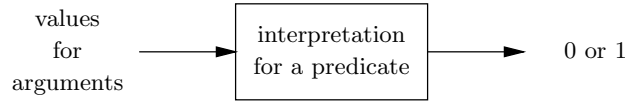
Interpretation for a predicate

Interpretation for an expression   Now we can define an *interpretation* for an expression E to be

---

[4] Unlike the relations discussed in Chapter 8, the relation that is the interpretation of a predicate may have an infinite set of tuples.

1.   A nonempty domain D, including any constants appearing in E,

2.   An interpretation for each predicate p appearing in E, and

3.   A value in D for each of the free variables of E, if any.

An interpretation and an "interpretation for a predicate" are illustrated in Fig. 14.6(a) and (b), respectively. Notice that interpretations play the role in predicate logic that is served by truth assignments in propositional logic.

(a) Interpretation for a predicate assigns truth values to tuples of values for the arguments.
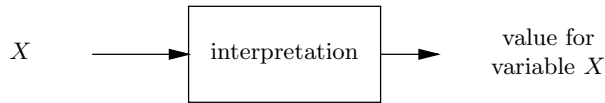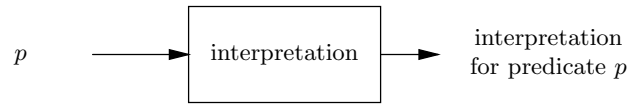
(b) Interpretation assigns a predicate interpretation to each predicate and a value to each variable (analogous to a truth assignment).
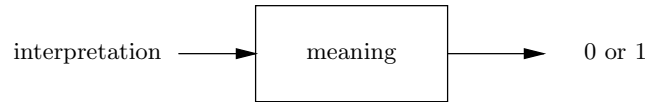
(c) Meaning of an expression assigns a truth value for each interpretation (analogous to a truth table).

Fig. 14.6. The meaning of expressions in predicate logic.

❖   **Example 14.10.** Consider the following expression of predicate logic:

$$p(X, Y) \rightarrow (\exists Z)\big(p(X, Z) \text{ AND } p(Z, Y)\big) \tag{14.5}$$

One possible interpretation for the predicate p, which we shall call interpretation I_1, is

1.   The domain D is the set of real numbers.

2. $p(U, V)$ is true whenever $U < V$. That is, the interpretation of p is the relation consisting of the infinite set of pairs (U, V) such that U and V are real numbers, and U is less than V.

Then (14.5) states that for any real numbers X and Y, if $X < Y$, then there is some Z lying strictly between X and Y; that is, $X < Z < Y$. For interpretation I_1, (14.5) is always true. If $X < Y$, we can pick $Z = (X + Y)/2$ — that is, the average of X and Y — and we can then be sure that $X < Z$ and $Z < Y$. If $X \geq Y$, then the left-hand side of the implication is false, so surely (14.5) is true.

   We can build an infinite number of interpretations for (14.5) based on the interpretation I_1 for predicate p, by picking any real numbers for the free variables X and Y. By what we just said, any of these interpretations for (14.5) will make (14.5) true.

   A second possible interpretation, I_2, for p is

1. D is the set of integers.

2. $p(U, V)$ is true if and only if $U < V$.

Now, we claim that (14.5) is true unless $Y = X + 1$. For if Y exceeds X by two or more, then Z can be selected to be $X + 1$. It will then be the case that $X < Z < Y$. If $Y \leq X$, then $p(X, Y)$ is false, and so (14.5) is again true. However, if $Y = X + 1$, then $p(X, Y)$ is true, but there is no integer Z lying strictly between X and Y. Thus for every integer Z, either $p(X, Z)$ or $p(Z, Y)$ will be false, and the right-hand side of the implication — that is, $(\exists Z)\big(p(X, Z) \text{ AND } p(Z, Y)\big)$ — is not true.

   We can extend I_2 to an interpretation for (14.5) by assigning integers to the free variables X and Y. The analysis above shows that (14.5) will be true for any such interpretation, except for those in which $Y = X + 1$.

   Our third interpretation for p, I_3, is abstract, without a common meaning in mathematics like those possessed by interpretations I_1 and I_2:

1. D is the set of three symbols a, b, c.

2. $p(U, V)$ is true if UV is one of the six pairs

   $$aa, ab, ba, bc, cb, cc$$

   and false for the other three pairs: ac, bb, and ca.

Then it happens that (14.5) is true for each of the nine pairs XY. In each case, either $p(X, Y)$ is false, or there is a Z that makes the right side of (14.5) true. The nine cases are enumerated in Fig. 14.7. We may extend I_3 to an interpretation for (14.5) in nine ways, by assigning any combination of a, b, and c to the free variables X and Y. Each of these interpretations imparts the value true to (14.5). ◆

Meaning of Expressions

Recall that the meaning of an expression in propositional logic is a function from the truth assignments to truth values 0 and 1, as was illustrated in Fig. 14.5(b). That is, a truth assignment states all that there is to know about the values of the atomic operands of the expression, and the expression then evaluates to 0 or 1. Similarly, in predicate logic, the meaning of an expression is a function that takes an interpretation, which is what we need to evaluate the atomic operands, and returns 0 or 1. This notion of meaning was illustrated in Fig. 14.6(c).

X	Y	Why true
a	a	$Z = a$ or b
a	b	$Z = a$
a	c	$p(a, c)$ false
b	a	$Z = a$
b	b	$p(b, b)$ false
b	c	$Z = c$
c	a	$p(c, a)$ false
c	b	$Z = c$
c	c	$Z = b$ or c

Fig. 14.7. Value of (14.5) under interpretation I_3.

❖ **Example 14.11.** Consider the expression (14.5) from Example 14.10. The free
variables of (14.5) are X and Y. If we are given interpretation I_1 of Example 14.10
for p (p is $<$ on reals), and we are given values $X = 3.14$ and $Y = 3.5$, then the
value of (14.5) is 1. In fact, with interpretation I_1 for p and any values for X and
Y, the expression has value 1, as was discussed in Example 14.10. The same is true
of interpretation I_3 for p; any values for X and Y chosen from the domain $\{a, b, c\}$.
gives (14.5) the value 1.

On the other hand, if we are given interpretation I_2 (p is $<$ on integers) and
values $X = 3$ and $Y = 4$, then (14.5) has value 0 as we discussed in Example 14.10.
If we have interpretation I_2 and values $X = 3$ and $Y = 5$ for the free variables, then
(14.5) has the value 1. ❖

To complete the definition of "meaning" for an expression, we must formally
define how the truth values for atomic operands are translated to a truth value for
the expression as a whole. We have been using our intuition previously, based on
our understanding of how the logical connectives of propositional logic work and our
intuition regarding quantifiers. The formal definition of the value of an expression,
given an interpretation I with domain D, is a structural induction on the expression
tree for the given logical expression E.

BASIS. If the expression tree is a leaf, then E is an atomic formula $p(X_1, \ldots, X_k)$.
The X_i's are all either constants or free variables of expression E. Interpretation I
gives us a value for each of the variables, and so we have values for all arguments of
p. Likewise, I tells us whether p, with those values as arguments, is true or false.
That truth value is the value of the expression E.

INDUCTION. Now, we must assume that we are given an expression E whose
expression tree has an operator at the root. There are several cases, depending on
what the operator at the root of E is.

First, consider the case where E is of the form E_1 AND E_2; that is, the operator
at the root is AND. The inductive hypothesis may be applied to the subexpressions
E_1 and E_2. We can thus evaluate E_1 under interpretation I.[5] Likewise, we can

---

[5] Strictly speaking, we must throw away from I the interpretation for any predicate p that

evaluate E_2 under the interpretation I. If both evaluate to 1, then E evaluates to 1; otherwise, E evaluates to 0.

The induction for other logical operators like OR or NOT is carried out the same way. For OR, we evaluate the two subexpressions and produce value 1 if either subexpression produces value 1; for NOT we evaluate the one subexpression and produce the negation of the value of that expression, and so on for the other operators of propositional logic.

Now suppose E is of the form $(\exists X)E_1$. The root operator is the existential quantifier, and we can apply the inductive hypothesis to the subexpression E_1. The predicates in E_1 all appear in E, and the free variables in E_1 are the free variables of E, plus (possibly) X.[6] Hence, we may construct, for each value v in the domain D, an interpretation for E_1 that is I, with the addition of the assignment of value v to variable X; call this interpretation J_v. We ask, for each value v, whether E_1 is true under interpretation J_v. If there is at least one such value v, then we say that $E = (\exists X)E_1$ is true; otherwise, we say E is false.

Last, suppose that E is of the form $(\forall X)E_1$. Again, the inductive hypothesis applies to E_1. Now we ask whether for every value v in the domain D, E_1 is true under the interpretation J_v. If so, we say E has value 1; if not, E has value 0.

✦   **Example 14.12.** Let us evaluate expression (14.5) with the interpretation I_2 for p ($<$ on the integers) and the values 3 and 7 for free variables X and Y, respectively. The expression tree for (14.5) is shown in Fig. 14.8. We observe that the operator at the root is $\rightarrow$. We did not cover this case explicitly, but the principle should be clear. The entire expression can be written as $E_1 \rightarrow E_2$, where E_1 is $p(X,Y)$, and E_2 is $(\exists Z)(p(X,Z) \text{ AND } p(Z,Y))$. Because of the meaning of $\rightarrow$, the entire expression (14.5) is true except in the case that E_1 is true and E_2 is false.
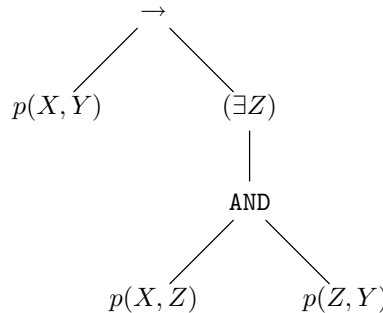

Fig. 14.8. Expression tree for (14.5).

E_1, which is $p(X,Y)$, is easy to evaluate. Since $X = 3$, $Y = 7$, and $p(X,Y)$ is true if and only if $X < Y$, we conclude that E_1 is true. To evaluate E_2 is more

---

appears in E but not E_1. Also, we must drop the value for any free variable that appears in E but not E_1. However, there is no conceptual difficulty if we include in an interpretation additional information that is not used.

[6] Technically, E_1 might not have any free occurrences of X, even though we apply a quantifier involving X to E_1. In that case, the quantifier may as well not be there, but we have not prohibited its presence.

Can we Compute Values of Expressions?

You may be suspicious of our definition as to when an expression E has value 1, in the cases where E is $(\exists X)E_1$ or $(\forall X)E_1$. If domain D is infinite, the test we have proposed, to evaluate E_1 under each interpretation J_v, need not have an algorithm for its execution. Essentially, we are asked to execute function

```
for (each v in D)
    if (E_1 is true under interpretation J_v)
        return TRUE;
return FALSE;
```

for an existential quantifier, and function

```
for (each v in D)
    if (E_1 is false under interpretation J_v)
        return FALSE;
return TRUE;
```

for a universal quantifier.

While the intent of these programs should be apparent, they are not algorithms, since when the domain D is infinite, we go around the loop an infinite number of times. However, although we may not be able to tell whether E is true or false, we are nevertheless offering the correct definition of when E is true; that is, we are ascribing the intended meaning to the quantifiers $\forall$ and $\exists$. In many practical and useful situations, we shall be able to tell whether E is true or false. In other situations, typically involving transformation of expressions into equivalent forms, we shall see that it doesn't matter whether E is true or false. We shall be able to reason that two expressions are equivalent from the definitions of their values, without knowing whether a value v that makes a subexpression like E_1 true exists.

---

difficult. We must consider all possible values of v for Z, to see if there is at least one value that makes $p(X,Z)$ AND $p(Z,Y)$ true. For example, if we try $Z = 0$, then $p(Z,Y)$ is true, but $p(X,Z)$ is false, since $X = 3$ is not less than Z.

If we think about the matter, we see that to make $p(X,Z)$ AND $p(Z,Y)$ true, we need a value of v such that $3 < v$ [so $p(X,Z)$ will be true] and such that $v < 7$ [so $p(Z,Y)$ will be true]. For example, $v = 4$ makes $p(X,Z)$ AND $p(Z,Y)$ true and therefore shows that E_2, or $(\exists Z)\big(p(X,Z) \text{ AND } p(Z,Y)\big)$, is true for the given interpretation.

We now know that both E_1 and E_2 are true. Since $E_1 \to E_2$ is true when both E_1 and E_2 are true, we conclude that (14.5) has value 1 for the interpretation in which predicate p has the interpretation I_2, $X = 3$, and $Y = 7$. ◆

EXERCISES

14.5.1: For each of the following expressions, give one interpretation that makes it true and one interpretation that makes it false.

a) $(\forall X)(\exists Y)(loves(X, Y))$
b) $p(X) \rightarrow \text{NOT } p(X)$
c) $(\exists X)p(X) \rightarrow (\forall X)p(X)$
d) $\big(p(X, Y) \text{ AND } p(Y, Z)\big) \rightarrow p(X, Z)$

14.5.2: Explain why every interpretation makes the expression $p(X) \rightarrow p(X)$ true.

❖❖ 14.6   Tautologies

Recall that in propositional logic, we call an expression a tautology if for every truth assignment, the value of the expression is 1. The same idea holds true in predicate logic. An expression E is called a *tautology* if for every interpretation of E, the value of E is 1.

❖ **Example 14.13.** As in propositional logic, it is rare for a "random" expression of predicate logic to be a tautology. For example, the expression (14.5), or

$$p(X, Y) \rightarrow (\exists Z)\big(p(X, Z) \text{ AND } p(Z, Y)\big)$$

which we studied in Example 14.10, is always true under some interpretations for predicate p, but there are interpretations such as I_2 of Example 14.10 — p is $<$ on the integers — for which this expression is not always true (e.g., it is false for $X = 1$ and $Y = 2$). Thus the expression is not a tautology.

An example of a tautology is the expression

$$q(X) \text{ OR NOT } q(X)$$

Here, it does not matter what interpretation we use for predicate q, or what value we assign to the free variable X. If our choice of interpretation makes $q(X)$ true, then the expression is true. If our choice makes $q(X)$ false, then it must make NOT $q(X)$ true, and again the expression is true. ❖

The Substitution Principle

The tautologies of propositional logic are a rich source of tautologies for predicate logic. The principle of substitution, which we introduced in Section 12.7, states that we can take any tautology of propositional logic, make any substitution for the propositional variables, and the result will still be a tautology. This principle still holds true if we allow the substitution of expressions of predicate logic for the propositional variables. For example, the tautology $q(X) \text{ OR NOT } q(X)$, mentioned in Example 14.13, is the substitution of the expression $q(X)$ for propositional variable p in the tautology $p \text{ OR NOT } p$.

The reason the principle of substitution holds true when expressions of predicate logic are substituted for propositional variables is much the same as the reason it holds true when propositional expressions are substituted. When we replace all occurrences of a propositional variable such as p by an expression like $q(X)$, we know that for any interpretation, the value of the substituted expression will be the same wherever it occurs. Since the original expression of propositional logic, into which the substitution is made, is a tautology, it is always true when a consistent substitution of 0 or a consistent substitution of 1 is made for one of its propositional variables.

For example, in the expression $q(X)$ OR NOT $q(X)$, no matter what the interpretation of q or the value of X, $q(X)$ is either true or false. Thus, either the expression becomes 1 OR NOT 1 or it becomes 0 OR NOT 0, both of which evaluate to 1.

Equivalence of Expressions

As in propositional logic, we can define two expressions, E and F, of predicate logic to be equivalent if $E \equiv F$ is a tautology. The "principle of substitution of equals for equals," also introduced in Section 12.7, continues to hold when we have equivalent expressions of predicate logic. That is, if E_1 is equivalent to E_2, then we may substitute E_2 for E_1 in any expression F_1, and the resulting expression F_2 will be equivalent; that is, $F_1 \equiv F_2$.

◆ **Example 14.14.** The commutative law for AND says $(p$ AND $q) \equiv (q$ AND $p)$. We might substitute $p(X)$ for p and $q(Y, Z)$ for q, giving us the tautology of predicate logic

$$\big(p(X) \text{ AND } q(Y, Z)\big) \equiv \big(q(Y, Z) \text{ AND } p(X)\big)$$

Thus the expressions $p(X)$ AND $q(Y, Z)$ and $q(Y, Z)$ AND $p(X)$ are equivalent. Now, if we have an expression like $\big(p(X)$ AND $q(Y, Z)\big)$ OR $q(X, Y)$, we can substitute $q(Y, Z)$ AND $p(X)$ for $p(X)$ AND $q(Y, Z)$, to produce another expression,

$$\big(q(Y, Z) \text{ AND } p(X)\big) \text{ OR } q(X, Y)$$

and know that

$$\Big(\big(p(X) \text{ AND } q(Y, Z)\big) \text{ OR } q(X, Y)\Big) \equiv \Big(\big(q(Y, Z) \text{ AND } p(X)\big) \text{ OR } q(X, Y)\Big)$$

There are more subtle cases of equivalent expressions in predicate logic. Normally, we would expect the equivalent expressions to have the same free variables and predicates, but there are some cases in which the free variables and/or predicates can be different. For example, the expression

$$\big(p(X) \text{ OR NOT } p(X)\big) \equiv \big(q(Y) \text{ OR NOT } q(Y)\big)$$

is a tautology, simply because both sides of the $\equiv$ are tautologies, as we argued in Example 14.13. Thus in the expression $p(X)$ OR NOT $p(X)$ OR $q(X)$ we may substitute $q(Y)$ OR NOT $q(Y)$ for $p(X)$ OR NOT $p(X)$, to deduce the equivalence

$$\big(p(X) \text{ OR NOT } p(X) \text{ OR } q(X)\big) \equiv \big(q(Y) \text{ OR NOT } q(Y) \text{ OR } q(X)\big)$$

Since the left-hand side of the $\equiv$ is a tautology, we can also infer that

$$q(Y) \text{ OR NOT } q(Y) \text{ OR } q(X)$$

is a tautology. ◆

EXERCISES

14.6.1: Explain why each of the following are tautologies. That is, what expression(s) of predicate logic did we substitute into which tautologies of propositional logic?

a) $\big(p(X) \text{ OR } q(Y)\big) \equiv \big(q(Y) \text{ OR } p(X)\big)$

b) $\big(p(X, Y) \text{ AND } p(X, Y)\big) \equiv p(X, Y)$

c) $\big(p(X) \rightarrow \text{FALSE}\big) \equiv \text{NOT } p(X)$

❖❖ 14.7   Tautologies Involving Quantifiers

Tautologies of predicate logic that involve quantifiers do not have direct counterparts in propositional logic. This section explores these tautologies and shows how they can be used to manipulate expressions. The main result of this section is that we can convert any expression into an equivalent expression with all the quantifiers at the beginning.

Variable Renaming

In C, we can change the name of a local variable, provided we change all uses of that local variable consistently. Analogously, one can change the variable used in a quantifier, provided we also change all occurrences of that variable bound to the quantifier. Also as in C, we must be careful which new variable name we pick, because if we choose a name that is defined outside the function in question, then we may change the meaning of the program, thereby committing a serious error.

Bearing in mind this kind of renaming, we can consider the following type of equivalence and conditions under which it is a tautology.

$$(QX)E \equiv (QY)E' \tag{14.6}$$

where E' is E with all occurrences of X that are bound to the explicitly shown quantifier (QX) replaced by Y. We claim that (14.6) is a tautology, provided no occurrence of Y is free in E. To see why, consider any interpretation I for $(QX)E$ (or equivalently, for $(QY)E'$, since the free variables and predicates of either quantified expression are the same). If I, extended by giving X the value v, makes E true, then I with the value v for Y will make E' true. Conversely, if extending I by using v for X makes E false, then extending I with v for Y makes E' false.

If quantifier Q is $\exists$, then should there be a value v for X that makes E true, there will be a value, namely v, for Y that makes E' true, and conversely. If Q is $\forall$, then all values of X will make E true if and only if all values of Y make E' true. Thus, for either quantifier, $(QX)E$ is true under any given interpretation I if and only if $(QY)E'$ is true under the same interpretation, showing that

$$(QX)E \equiv (QY)E'$$

is a tautology.

❖   **Example 14.15.** Consider the expression

$$\big((\exists X)p(X, Y)\big) \text{ OR NOT } \big((\exists X)p(X, Y)\big) \tag{14.7}$$

Making Quantified Variables Unique

An interesting consequence of (14.6) is that we can always turn any expression E of predicate logic into an equivalent expression in which no two quantifiers use the same variable, and also, no quantifier uses a variable that is also free in E. Such an expression is called *rectified*.

Rectified expression

In proof, we may start with the tautology $E \equiv E$. Then, we use (14.6) on the occurrence of E on the right to rename, in turn, each quantified variable by a new variable not used elsewhere in E. The result is an expression $E \equiv E'$, where all quantifiers (QX) of E' involve distinct X's, and these X's do not appear free in E or E'. By the law of transitivity of equivalence for propositional logic, $E \equiv E'$ is a tautology; that is, E and E' are equivalent expressions.

which happens to be a tautology. We shall show how to rename one of the two X's, to form another tautology with distinct variables used in the two quantifiers.

If we let E in (14.6) be $p(X, Y)$, and we choose variable Z to play the role of Y in (14.6), then we have the tautology $\big((\exists X)p(X, Y)\big) \equiv \big((\exists Z)p(Z, Y)\big)$. That is, to construct the expression E' we substitute Z for X in $E = p(X, Y)$, to obtain $p(Z, Y)$. Thus we can substitute "equals for equals," replacing the first occurrence of $(\exists X)p(X, Y)$ in (14.7) by $(\exists Z)p(Z, Y)$, to obtain the expression

$$\big((\exists Z)p(Z, Y)\big) \text{ OR NOT } \big((\exists X)p(X, Y)\big).$$

This expression is equivalent to (14.7), and therefore is also a tautology.

Note that we could also replace X in the second half of (14.7) by Z; it doesn't matter whether or not we do so, because the two quantifiers define distinct and unrelated variables, each of which was named X in (14.7). However, we should understand that it is not permissible to replace either occurrence of $\exists X$ by $\exists Y$, because Y is free in each of the subexpressions $p(X, Y)$.

That is, $\big((\exists X)p(X, Y)\big) \equiv \big((\exists Y)p(Y, Y)\big)$ is not an instance of (14.6) that is a tautology, because Y is free in the expression $p(X, Y)$. To see that it is not a tautology, let p be interpreted as $<$ on integers. Then for any value of the free variable Y, say $Y = 10$, the expression $(\exists X)p(X, Y)$ is true, because we can let $X = 9$, for example. Yet the right side of the equivalence, $(\exists Y)p(Y, Y)$ is false, because no integer is strictly less than itself.

Similarly, it is not permissible to substitute $(\exists Y)p(Y, Y)$ for the first instance of $(\exists X)p(X, Y)$ in (14.7). The resulting expression,

$$\big((\exists Y)p(Y, Y)\big) \text{ OR NOT } \big((\exists X)p(X, Y)\big) \tag{14.8}$$

can also be seen not to be a tautology. Again, let the interpretation of p be $<$ on the integers, and let, for instance, the value of the free variable Y be 10. Note that in (14.8), the first two occurrences of Y, in $p(Y, Y)$, are bound occurrences, bound to the quantifier $(\exists Y)$. Only the last occurrence of Y, in $p(X, Y)$, is free. Then $(\exists Y)p(Y, Y)$ is false for this interpretation, because no value of Y is less than itself. On the other hand, $(\exists X)p(X, Y)$ is true when $Y = 10$ (or any other integer, for that matter), and so NOT $\big((\exists X)p(X, Y)\big)$ is false. As a result, (14.8) is false for this interpretation. ❖

Universal Quantification of Free Variables

An expression with free variables can only be a tautology if the same expression with its free variables universally quantified is a tautology. Formally, for every tautology T and variable X, $(\forall X)T$ is also a tautology. Technically, it does not matter whether or not X appears free in T.

To see why $(\forall X)T$ is a tautology, let $Y_1, \ldots, Y_k$ be the free variables of T; X may or may not be one of them. First, suppose that $X = Y_1$. We need to show that for every interpretation I, $(\forall X)T$ is true. Equivalently, we need to show that for every value v in the domain of I, the interpretation J_v formed from I by giving X the value v makes T true. But T is a tautology, and so every interpretation makes it true.

If X is one of the other free variables Y_i of T, the argument that $(\forall X)T$ is a tautology is essentially the same. If X is none of the Y_i's, then its value doesn't affect the truth or falsehood of T. Thus T is true for all X, simply because T is a tautology.

Closed Expressions

An interesting consequence is that for tautologies, we can assume there are no free variables. We can apply the preceding transformation to universally quantify one free variable at a time. An expression with no free variables is called a *closed* expression.

❖ **Example 14.16.** We know that $p(X, Y)$ OR NOT $p(X, Y)$ is a tautology. We may add universal quantifiers for the free variables X and Y, to get the tautology

$$(\forall X)(\forall Y)\big(p(X, Y) \text{ OR NOT } p(X, Y)\big)$$

❖

Moving Quantifiers Through NOT

There is an infinite version of DeMorgan's law that lets us replace $\forall$ by $\exists$ or vice versa, just as the "normal" DeMorgan's laws allow us to switch between AND and OR, while moving through a NOT. Suppose that we have an expression like

$$\text{NOT } \big((\forall X)p(X)\big)$$

If the domain of values is finite, say $v_1, \ldots, v_n$, then we could think of this expression as NOT $\big(p(v_1) \text{ AND } p(v_2) \text{ AND } \cdots \text{ AND } p(v_n)\big)$. We could then apply DeMorgan's law to rewrite this expression as NOT $p(v_1)$ OR NOT $p(v_2)$ OR $\cdots$ OR NOT $p(v_n)$. On the assumption of a finite domain, this expression is the same as $(\exists X)\big(\text{NOT } p(X)\big)$ — that is, for some value of X, $p(X)$ is false.

In fact, this transformation does not depend on the finiteness of the domain; it holds for every possible interpretation. That is, the following equivalence is a tautology for any expression E.

$$\Big(\text{NOT } \big((\forall X)E\big)\Big) \equiv \big((\exists X)(\text{NOT } E)\big) \tag{14.9}$$

Informally, (14.9) says that E fails to be true for all X exactly when there is some value of X that makes E false.

There is a similar tautology that lets us push a NOT inside an existential quantifier.

$$\Big(\text{NOT}\left((\exists X)E\right)\Big) \equiv \left((\forall X)(\text{NOT } E)\right) \tag{14.10}$$

Informally, there does not exist an X that makes E true exactly when E is false for all X.

❖ **Example 14.17.** Consider the tautology

$$(\forall X)p(X) \text{ OR NOT}\left((\forall X)p(X)\right) \tag{14.11}$$

which we obtain by the principle of substitution from the tautology of propositional logic p OR NOT p. We may use (14.9) with $E = p(X)$, to replace NOT $\left((\forall X)p(X)\right)$ in (14.11) by $(\exists X)\left(\text{NOT } p(X)\right)$, to yield the tautology

$$(\forall X)p(X) \text{ OR } (\exists X)\left(\text{NOT } p(X)\right).$$

That is, either $p(X)$ is true for all X, or there exists some X for which $p(X)$ is false. ❖

Moving Quantifiers Through AND and OR

Laws (14.9) and (14.10), when applied from left to right, have the effect of moving a quantifier outside a negation, "inverting" the quantifier as we do so, that is, swapping $\forall$ for $\exists$ and vice versa. Similarly, we can move quantifiers outside of AND or OR, but we must be careful not to change the binding of any variable occurrence. Also, we do not invert quantifiers moving through AND or OR. The expressions of these laws are

$$\left(E \text{ AND } (QX)F\right) \equiv (QX)(E \text{ AND } F) \tag{14.12}$$

$$\left(E \text{ OR } (QX)F\right) \equiv (QX)(E \text{ OR } F) \tag{14.13}$$

where E and F are any expressions, and Q is either quantifier. However, we require that X not be free in E.

Since AND and OR are commutative, we can also use (14.12) and (14.13) to move quantifiers attached to the left operand of AND or OR. For example, a form of tautology that follows from (14.12) and the commutativity of AND is

$$\left((QX)E \text{ AND } F\right) \equiv (QX)(E \text{ AND } F)$$

Here, we require that X is not free in F.

❖ **Example 14.18.** Let us transform the tautology developed in Example 14.17, that is,

$$(\forall X)p(X) \text{ OR } (\exists X)\left(\text{NOT } p(X)\right)$$

so that the quantifiers are outside the expression. First, we need to rename the variable used by one of the two quantifiers. By law (14.6), we can replace the subexpression $(\exists X)\ \texttt{NOT}\ p(X)$ by $(\exists Y)\ \texttt{NOT}\ p(Y)$, giving us the tautology

$$(\forall X)p(X)\ \texttt{OR}\ (\exists Y)\big(\texttt{NOT}\ p(Y)\big) \tag{14.14}$$

Now we can use (14.13), in its variant form where the quantifier on the left operand of the $\texttt{OR}$ is moved, to take the $\forall$ outside the $\texttt{OR}$. The resulting expression is

$$(\forall X)\Big(p(X)\ \texttt{OR}\ (\exists Y)\big(\texttt{NOT}\ p(Y)\big)\Big) \tag{14.15}$$

Expression (14.15) differs from (14.14) in form, but not in meaning; (14.15) states that for all values of X, at least one of the following holds:

1. $p(X)$ is true.

2.   There is some value of Y that makes $p(Y)$ false.

To see why (14.15) is a tautology, consider some value v for X. If the interpretation under consideration makes $p(v)$ true, then $p(X)\ \texttt{OR}\ (\exists Y)\big(\texttt{NOT}\ p(Y)\big)$ is true. If $p(v)$ is false, then in this interpretation, (2) must hold. In particular, when $Y = v$, $\texttt{NOT}\ p(Y)$ is true, and so $(\exists Y)\big(\texttt{NOT}\ p(Y)\big)$ is true.

Finally, we can apply (14.13) to move $\exists Y$ outside the $\texttt{OR}$. The expression that results is

$$(\forall X)(\exists Y)(p(X)\ \texttt{OR}\ \texttt{NOT}\ p(Y))$$

This expression also must be a tautology. Informally, it states that for every value of X, there exists some value of Y that makes $p(X)\ \texttt{OR}\ \texttt{NOT}\ p(Y)$ true. To see why, let v be a possible value of X. If $p(v)$ is true in given interpretation I, then surely

$$p(X)\ \texttt{OR}\ \texttt{NOT}\ p(Y)$$

is true, regardless of Y. If $p(v)$ is false in interpretation I, then we may pick v for Y, and $(\exists Y)\big(p(X)\ \texttt{OR}\ \texttt{NOT}\ p(Y)\big)$ will be true. ◆

Prenex Form

A consequence of the laws (14.9), (14.10), (14.12), and (14.13) is that, given any expression involving quantifiers and the logical operators $\texttt{AND}$, $\texttt{OR}$, and $\texttt{NOT}$, we can find an equivalent expression that has all its quantifiers on the outside (at the top of the expression tree). That is, we can find an equivalent expression of the form

$$(Q_1 X_1)(Q_2 X_2)\cdots(Q_k X_k)E \tag{14.16}$$

Quantifier-free expression

where $Q_1, \ldots, Q_k$ each stand for one of the quantifiers $\forall$ or $\exists$, and the subexpression E is *quantifier free* — that is, it has no quantifiers. The expression (14.16) is said to be in *prenex form*.

We can transform an expression into prenex form in two steps.

1.   Rectify the expression. That is, use law (14.6) to make each of the quantifiers refer to a distinct variable, one that appears neither in another quantifier nor free in the expression.

2.   Then, move each quantifier through $\texttt{NOT}$'s by laws (14.9) and (14.10), through $\texttt{AND}$'s by (14.12), and through $\texttt{OR}$'s by (14.13).

Programs in Prenex Form

In principle, we can put a C program in "prenex form," if we rename all local variables so that they are distinct, and then move their declarations into the main program. We generally don't want to do that; we prefer to declare variables locally, so that we don't have to worry, for example, about inventing different names for a variable i used as a loop index in ten different functions. For logical expressions, there is often a reason to put expressions in prenex form, although the matter is beyond the scope of this book.

✦   **Example 14.19.** Examples 14.17 and 14.18 were examples of this process. We started in Example 14.17 with the expression $(\forall X)p(X)$ OR NOT $\big((\forall X)p(X)\big)$. By moving the second $\forall$ through the NOT, we obtained the expression

$$(\forall X)p(X) \text{ OR } (\exists X)\big(\text{NOT } p(X)\big)$$

with which we started in Example 14.18. We then renamed the second use of X, which we could (and should) have done initially. By moving the two quantifiers through the OR, we obtained $(\forall X)(\exists Y)\big(p(X) \text{ OR NOT } p(Y)\big)$, which is in prenex form. ✦

Note that expressions involving logical operators other than AND, OR, and NOT can also be put in prenex form. Every logical operator can be written in terms of AND, OR, and NOT, as we learned in Chapter 12. For example, $E \rightarrow F$ can be replaced by NOT E OR F. If we write each logical operator in terms of AND, OR, and NOT, then we are able to apply the transformation just outlined to find an equivalent expression in prenex form.

Reordering Quantifiers

Our final family of tautologies is derived by noting that in applying a universal quantifier to two variables, the order in which we write the quantifiers does not matter. Similarly, we can write two existential quantifiers in either order. Formally, the following are tautologies.

$$(\forall X)(\forall Y)E \equiv (\forall Y)(\forall X)E \tag{14.17}$$

$$(\exists X)(\exists Y)E \equiv (\exists Y)(\exists X)E \tag{14.18}$$

Note that by (14.17), we can permute any string of $\forall$'s, $(\forall X_1)(\forall X_2)(\cdots)(\forall X_k)$ into whatever order we choose. In effect, (14.17) is the commutative law for $\forall$. Analogous observations hold for law (14.18), which is the commutative law for $\exists$.

EXERCISES

14.7.1: Transform the following expressions into rectified expressions, that is, expressions for which no two quantifier occurrences share the same variable.

a) $(\exists X)\Big(\big(\text{NOT } p(X)\big) \text{ AND } \big((\exists Y)(p(Y)) \text{ OR } \big((\exists X)\big(q(X, Z)\big)\big)\big)\Big)$

b) $(\exists X)\big((\exists X)p(X) \text{ OR } (\exists X)q(X) \text{ OR } r(X)\big)$

14.7.2: Turn the following into closed expressions by universally quantifying each of the free variables. If necessary, rename variables so that no two quantifier occurrences use the same variable.

a) $p(X, Y) \text{ AND } (\exists Y)q(Y)$
b) $(\forall X)\big(p(X, Y) \text{ OR } (\exists X)p(Y, X)\big)$

14.7.3*: Does law (14.12) imply that $p(X, Y) \text{ AND } (\forall X)q(X)$ is equivalent to

$$(\forall X)\big(p(X, Y) \text{ AND } q(X)\big)$$

Explain your answer.

14.7.4: Transform the expressions of Exercise 14.7.1 into prenex form.

14.7.5*: Show how to move quantifiers through an $\rightarrow$ operator. That is, turn the expression $\big((Q_1 X)E\big) \rightarrow \big((Q_2 Y)F\big)$ into a prenex form expression. What constraints on free variables in E and F do you need?

14.7.6: We can use tautologies (14.9) and (14.10) to move NOT's inside quantifiers as well as to move them outside. Using these laws, plus DeMorgan's laws, we can move all NOT's so they apply directly to atomic formulas. Apply this transformation to the following expressions.

a) $\text{NOT } \big((\forall X)(\exists Y)p(X, Y)\big)$
b) $\text{NOT } \Big((\forall X)\big(p(X) \text{ OR } (\exists Y)q(X, Y)\big)\Big)$

14.7.7*: Is it true that E is a tautology whenever $(\exists X)E$ is a tautology?

❖❖ 14.8 Proofs in Predicate Logic

In this chapter and the next, we shall discuss proofs in predicate logic. We do not, however, extend the resolution method of Section 12.11 to predicate logic, although it can be done. In fact, resolution is extremely important for many systems that use predicate logic. The mechanics of proofs were introduced in Section 12.10. Recall that in a proof of propositional logic we are given some expressions $E_1, E_2, \ldots, E_k$ as hypotheses, or "axioms," and we construct a sequence of expressions (lines) such that each expression either

1. Is one of the E_i's, or

2. Follows from zero or more of the previous expressions by some rule of inference.

Rules of inference must have the property that, whenever we are allowed to add F to the list of expressions because of the presence of $F_1, F_2, \ldots, F_n$ on the list,

$$(F_1 \text{ AND } F_2 \text{ AND } \cdots \text{ AND } F_n) \rightarrow F$$

is a tautology.

Proofs in predicate logic are much the same. Of course, the expressions that are hypotheses and lines of the proof are expressions of predicate logic, not propositional logic. Moreover, it does not make sense to have, in one expression, free variables that bear a relationship to a free variable of the same name in another expression. Thus we shall require that the hypotheses and lines of the proof be closed formulas.

Implicit Universal Quantifiers

However, it is conventional to write expressions in proofs without explicitly showing the outermost universal quantifiers. For example, consider the expression in Example 14.3,

$$\big(csg(\text{``CS101''}, S, G) \text{ AND } snap(S, \text{``C.Brown''}, A, P)\big) \to answer(G) \quad (14.19)$$

Expression (14.19) might be one of the hypotheses in a proof. In Example 14.3, we saw it intuitively as a definition of the predicate *answer*. We might use (14.19) in a proof of, say, $answer(\text{``A''})$ — that is, C. Brown received an A in course CS101.

In Example 14.3 we explained the meaning of (14.19) by saying that for all values of S, G, A, and P, if student S received grade G in CS101 — that is, if $csg(\text{``CS101''}, S, G)$ is true — and student $S$ has name "C. Brown," address $A$, and phone $P$ — that is, if $snap(S, \text{``C.Brown''}, A, P)$ is true — then G is an answer (i.e., $answer(G)$ is true). In that example, we did not have the formal notion of quantifiers. However, now we see that what we really want to assert is

$$(\forall S)(\forall G)(\forall A)(\forall P)\Big(\big((csg(\text{``CS101''}, S, G) \text{ AND } snap(S, \text{``C.Brown''}, A, P)\big)$$
$$\to answer(G)\Big)$$

Because it is frequently necessary to introduce a string of universal quantifiers around an expression, we shall adopt the shorthand notation $(\forall *)E$ to mean a string of quantifiers $(\forall X_1)(\forall X_2)(\cdots)(\forall X_k)E$, where $X_1, X_2, \ldots, X_k$ are all the free variables of expression E. For example, (14.19) could be written

$$(\forall *)\Big(\big(csg(\text{``CS101''}, S, G) \text{ AND } snap(S, \text{``C.Brown''}, A, P)\big) \to answer(G)\Big)$$

However, we shall continue to refer to variables that are free in E as "free" in $(\forall *)E$. This use of the term "free" is strictly incorrect, but is quite useful.

Substitution for Variables as an Inference Rule

In addition to the inference rules discussed in Chapter 12 for propositional logic, such as modus ponens, and substitution of equals for equals in a previous line of the proof, there is an inference rule involving substitution for variables that is quite useful for proofs in predicate logic. If we have asserted an expression E, either as a hypothesis or as a line of a proof, and E' is formed from E by substituting variables or constants for some of the free variables of E, then $E \to E'$ is a tautology, and we may add E' as a line of the proof. It is important to remember that we cannot substitute for bound variables of E, only for the free variables of E.

Formally, we can represent a substitution for variables by a function *sub*. For each free variable X of E, we may define $sub(X)$ to be some variable or some constant. If we do not specify a value for $sub(X)$, then we shall assume that $sub(X) = X$ is intended. If E is any expression of predicate logic, the expression $sub(E)$ is E with all free occurrences of any variable X replaced by $sub(X)$.

Expressions in Proofs

Remember that when we see an expression E in a proof, it is really short for the expression $(\forall*)E$. Note that $E \equiv (\forall*)E$ is generally not a tautology, and so we are definitely using one expression to stand for a different expression.

It is also helpful to remember that when E appears in a proof, we are not asserting that $(\forall*)E$ is a tautology. Rather, we are asserting that $(\forall*)E$ follows from the hypotheses. That is, if $E_1, E_2, \ldots, E_n$ are the hypotheses, and we correctly write proof line E, then we know

$$\big((\forall*)E_1 \text{ AND } (\forall*)E_2 \text{ AND } \cdots \text{ AND } (\forall*)E_n\big) \rightarrow (\forall*)E$$

is a tautology.

The *law of variable substitution* says that $E \rightarrow sub(E)$ is a tautology. Thus, if E is a line of a proof, we may add $sub(E)$ as a line of the same proof.

✦ **Example 14.20.** Consider the expression (14.19)

$$\big(csg(\text{``CS101''}, S, G) \text{ AND } snap(S, \text{``C.Brown''}, A, P)\big) \rightarrow answer(G)$$

as E. A possible substitution sub is defined by

$$sub(G) = \text{``B''}$$
$$sub(P) = S$$

That is, we substitute the constant "B" for the variable G and we substitute variable S for variable P. The variables S and A remain unchanged. The expression $sub(E)$ is

$$\big(csg(\text{``CS101''}, S, \text{``B''}) \text{ AND } snap(S, \text{``C.Brown''}, A, S)\big) \rightarrow answer(\text{``B''}) \qquad (14.20)$$

Informally, (14.20) says that if there is a student S who received a B in CS101, and the student's name is C. Brown, and the student's phone number and student ID are identical, then "B" is an answer.

Notice that (14.20) is a special case of the more general rule expressed by (14.19). That is, (14.20) only infers the correct answer in the case that the grade is B, and C. Brown, by a strange coincidence, has the same student ID and phone number; otherwise (14.20) infers nothing. ✦

✦ **Example 14.21.** The expression

$$p(X, Y) \text{ OR } (\exists Z)q(X, Z) \qquad (14.21)$$

has free variables X and Y, and it has bound variable Z. Recall that technically, (14.21) stands for the closed expression $(\forall*)\big(p(X, Y) \text{ OR } (\exists Z)q(X, Z)\big)$, and that here the $(\forall*)$ stands for quantification over the free variables X and Y, that is,

$$(\forall X)(\forall Y)\big(p(X, Y) \text{ OR } (\exists Z)q(X, Z)\big)$$

Substitution as Special-Casing

Example 14.20 is typical, in that whenever we apply a substitution *sub* to an expression E, what we get is a special case of E. If *sub* replaces variable X by a constant c, then the expression $sub(E)$ only applies when $X = c$, and not otherwise. If *sub* makes two variables become the same, then $sub(E)$ only applies in the special case that these two variables have the same value. Nonetheless, substitutions for variables are often exactly what we need to make a proof, because they allow us to apply a general rule in a special case, and they allow us to combine rules to make additional rules. We shall study this form of proof in the next section.

In (14.21), we might substitute $sub(X) = a$, and $sub(Y) = b$, yielding the expression $p(a, b)$ OR $(\exists Z)q(a, Z)$. This expression, which has no free variables because we chose to substitute a constant for each free variable, is easily seen to be a special case of (14.21); it states that either $p(a, b)$ is true or for some value of Z, $q(a, Z)$ is true. Formally,

$$\Big((\forall X)(\forall Y)\big(p(X, Y) \text{ OR } (\exists Z)q(X, Z)\big)\Big) \rightarrow \big(p(a, b) \text{ OR } (\exists Z)q(a, Z)\big)$$

is a tautology.

One might wonder what happened to the implied quantifiers in (14.21) when we substituted a and b for X and Y. The answer is that in the resulting expression, $p(a, b)$ OR $(\exists Z)q(a, Z)$, there are no free variables, and so the implied expression $(\forall *)(p(a, b)$ OR $(\exists Z)q(a, Z))$ has no prefix of universal quantifiers; that is,

$$p(a, b) \text{ OR } (\exists Z)q(a, Z)$$

stands for itself in this case. We do not replace $(\forall *)$ by $(\forall a)(\forall b)$, which makes no sense, since constants cannot be quantified. ❖

EXERCISES

14.8.1: Prove the following conclusions from hypotheses, using the inference rules discussed in Section 12.10, plus the variable-substitution rule just discussed. Note that you can use as a line of proof any tautology of either propositional or predicate calculus. However, try to restrict your tautologies to the ones enumerated in Sections 12.8, 12.9, and 14.7.

a)   From hypothesis $(\forall X)p(X)$ prove the conclusion $(\forall X)p(X)$ OR $q(Y)$.

b)   From hypothesis $(\exists X)p(X, Y)$ prove the conclusion NOT $\Big((\forall X)\big(\text{NOT } p(X, a)\big)\Big)$.

c)   From the hypotheses $p(X)$ and $p(X) \rightarrow q(X)$ prove the conclusion $q(X)$.

❖❖ 14.9 Proofs from Rules and Facts

Perhaps the simplest form of proof in predicate logic involves hypotheses that fall into two classes.

1.   *Facts*, which are ground atomic formulas.

2.   *Rules,* which are "if-then" expressions. An example is the query (14.19) about C. Brown's grade in CS101,

$$\bigl(csg(\text{“CS101”}, S, G) \text{ AND } snap(S, \text{“C.Brown”}, A, P)\bigr) \rightarrow answer(G)$$

which we discussed in Example 14.20. Rules consist of the AND of one or more atomic formulas on the left-hand side of the implication sign, and one atomic formula on the right-hand side. We assume that any variable appearing in the head also appears somewhere in the body.

Body, head, subgoal

The left-hand side (hypotheses) of a rule is called the *body*, and the right-hand side side is called the *head*. Any one of the atomic formulas of the body is called a *subgoal*. For instance, in (14.19), the rule repeated above, the subgoals are $csg(\text{“CS101”}, S, G)$ and $snap(S, \text{“C.Brown”}, A, P)$. The head is $answer(G)$.

The general idea behind the use of rules is that rules are general principles that we can apply to facts. We try to match the subgoals of the body of a rule to facts that are either given or already proven, by substituting for variables in the rule. When we can do so, the substitution makes the head a ground atomic formula, because we have assumed that each variable of the head appears in the body. We can add this new ground atomic formula to the collection of facts at our disposal for further proofs.

❖   **Example 14.22.**   One simple application of proofs from rules and facts is in answering queries as in the relational model discussed in Chapter 8. Each relation corresponds to a predicate symbol, and each tuple in the relation corresponds to a ground atomic formula with that predicate symbol and with arguments equal to the components of the tuple, in order. For example, from the Course-Student-Grade relation of Fig. 8.1, we would get the facts

$$csg(\text{“CS101”}, 12345, \text{“A”}) \qquad csg(\text{“CS101”}, 67890, \text{“B”})$$
$$csg(\text{“EE200”}, 12345, \text{“C”}) \qquad csg(\text{“EE200”}, 22222, \text{“B+”})$$
$$csg(\text{“CS101”}, 33333, \text{“A–”}) \qquad csg(\text{“PH100”}, 67890, \text{“C+”})$$

Similarly, from the Student-Name-Address-Phone relation of Fig. 8.2(a), we get the facts

$$snap(12345, \text{“C.Brown”}, \text{“12 Apple St.”}, 555\text{-}1234)$$
$$snap(67890, \text{“L.VanPelt”}, \text{“34 Pear Ave.”}, 555\text{-}5678)$$
$$snap(22222, \text{“P.Patty”}, \text{“56 Grape Blvd.”}, 555\text{-}9999)$$

To these facts, we might add the rule (14.19),

$$\bigl(csg(\text{“CS101”}, S, G) \text{ AND } snap(S, \text{“C.Brown”}, A, P)\bigr) \rightarrow answer(G)$$

to complete the list of hypotheses.

Suppose we want to show that $answer(\text{“A”})$ is true, that is, C. Brown gets an A in CS101. We could begin our proof with all of the facts and the rule, although in this case we only need the rule, the first csg fact and the first $snap$ fact. That is, the first three lines of the proof are

1. $\bigl(csg(\text{“CS101”}, S, G) \text{ AND } snap(S, \text{“C.Brown”}, A, P)\bigr) \rightarrow answer(G)$

2. $csg($ "CS101" $, 12345,$ "A" $)$

3. $snap(12345,$ "C.Brown" $,$ "12 Apple St." $, 555\text{-}1234)$

The next step is to combine the second and third lines, using the inference rule that says if E_1 and E_2 are lines of a proof then E_1 `AND` E_2 may be written as a line of the proof. Thus we have line

4. $csg($ "CS101" $, 12345,$ "A" $)$ `AND`
     $snap(12345,$ "C.Brown" $,$ "12 Apple St." $, 555\text{-}1234)$

Next, let us use the law of substitution for free variables to specialize our rule — line (1) — so that it applies to the constants in line (4). That is, we make the substitution

$$sub(S) = \text{``CS101''}$$
$$sub(G) = \text{``A''}$$
$$sub(A) = \text{``12 Apple St.''}$$
$$sub(P) = 555\text{-}1234$$

in (1) to obtain the line

5. $\big(csg($ "CS101" $, 12345,$ "A" $)$ `AND`
     $snap(12345,$ "C.Brown" $,$ "12 Apple St." $, 555\text{-}1234)\big)$
     $\rightarrow answer($ "A" $)$

Finally, modus ponens applied to (4) and (5) gives us the sixth and last line of the proof,

6. $answer($ "A" $)$. ◆

A Simplified Inference Rule

If we look at the proof of Example 14.22, we can observe the following strategy for building a proof from ground atomic formulas and logical rules.

1.   We select a rule to apply and we select a substitution that turns each subgoal into a ground atomic formula that either is a given fact, or is something we have already proved. In Example 14.22, we substituted 12345 for S, and so on. The result appeared as line (4) of Example 14.22.

2.   We create lines of the proof for each of the substituted subgoals, either because they are facts, or by inferring them in some way. This step appeared as lines (2) and (3) in Example 14.22.

3.   We create a line that is the `AND` of the lines corresponding to each of the substituted subgoals. This line is the body of the substituted rule. In Example 14.22, this step appeared as line (5).

4.   We use modus ponens, with the substituted body from (3) and the substituted rule from (1) to infer the substituted head. This step appeared as line (6) in Example 14.22.

We can combine these steps into a single inference rule, as follows. If there is a rule R among the hypotheses and a substitution sub such that in the substituted instance $sub(R)$, each of the subgoals is a line of the proof, then we may add the head of $sub(R)$ as a line of the proof.

Interpreting Rules

Rules, like all expressions that appear in proofs, are implicitly universally quantified. Thus we can read (14.19) as "for all S, G, A, and P, if $csg($ "CS101"$, S, G)$ is true, and $snap(S,$ "C.Brown"$, A, P)$ is true, then $answer(G)$ is true." However, we may treat variables that appear in the body, but not in the head, such as S, A, and P, as existentially quantified for the scope of the body. Formally, (14.19) is equivalent to

$$(\forall G)\Big((\exists S)(\exists A)(\exists P)\big(csg(\text{``CS101''}, S, G) \text{ AND } snap(S, \text{``C.Brown''}, A, P)\big)$$
$$\rightarrow answer(G)\Big)$$

That is, for all G, if there exists S, A, and P such that $csg($ "CS101"$, S, G)$ and $snap(S,$ "C.Brown"$, A, P)$ are both true, then $answer(G)$ is true.

This phrasing corresponds more closely to the way we think of applying a rule. It suggests that for each value of the variable or variables that appear in the head, we should try to find values of the variables appearing only in the body, that make the body true. If we find such values, then the head is true for the chosen values of its variables.

To see why we can treat variables that are local to the body as existentially quantified, start with a rule of the form $B \rightarrow H$, where B is the body, and H is the head. Let X be one variable that appears only in B. Implicitly, this rule is

$$(\forall *)(B \rightarrow H)$$

and by law (14.17), we can make the quantifier for X be the innermost, writing the expression as $(\forall *)(\forall X)(B \rightarrow H)$. Here, the $(\forall *)$ includes all variables but X. Now we replace the implication by its equivalent expression using NOT and OR, that is, $(\forall *)(\forall X)\big((\text{NOT } B) \text{ OR } H\big)$. Since X does not appear in H, we may apply law (14.13) in reverse, to make the $(\forall X)$ apply to NOT B only, as $(\forall *)\Big(\big((\forall X) \text{ NOT } B\big) \text{ OR } H\Big)$. Next, we use law (14.10) to move the $(\forall X)$ inside the negation, yielding

$$(\forall *)\Big(\big(\text{NOT } (\exists X)(\text{NOT NOT } B)\big) \text{ OR } H\Big)$$

or, after eliminating the double negation, $(\forall *)\Big(\big(\text{NOT } (\exists X)B\big) \text{ OR } H\Big)$. Finally, we restore the implication to get $(\forall *)\Big(\big((\exists X)B\big) \rightarrow H\Big)$.

✦ **Example 14.23.** In Example 14.22, rule R is (14.19), or

$$\big(csg(\text{``CS101''}, S, G) \text{ AND } snap(S, \text{``C.Brown''}, A, P)\big) \rightarrow answer(G)$$

The substitution sub is as given in that example, and the subgoals of $sub(R)$ are lines (2) and (3) in Example 14.22. By the new inference rule, we could write down line (6) of Example 14.22 immediately; we do not need lines (4) and (5). In fact, line (1), the rule R itself, can be omitted from the proof as long as it is a given hypothesis. ✦

❖   **Example 14.24.** For another example of how rules may be applied in proofs, let us consider the Course-Prerequisite relation of Fig. 8.2(b), whose eight facts can be represented by eight ground atomic formulas with predicate cp,

$$cp(\text{``CS101''}, \text{``CS100''}) \qquad cp(\text{``EE200''}, \text{``EE005''})$$
$$cp(\text{``EE200''}, \text{``CS100''}) \qquad cp(\text{``CS120''}, \text{``CS101''})$$
$$cp(\text{``CS121''}, \text{``CS120''}) \qquad cp(\text{``CS205''}, \text{``CS101''})$$
$$cp(\text{``CS206''}, \text{``CS121''}) \qquad cp(\text{``CS206''}, \text{``CS205''})$$

We might wish to define another predicate $before(X, Y)$ that means course Y must be taken before course X. Either Y is a prerequisite of X, a prerequisite of a prerequisite of X, or so on. We can define the notion "before" recursively, by saying

1.   If Y is a prerequisite of X, then Y comes before X.

2.   If X has a prerequisite Z, and Y comes before Z, then Y comes before X.

Rules (1) and (2) can be expressed as rules of predicate logic as follows.

$$cp(X, Y) \rightarrow before(X, Y) \tag{14.22}$$

$$\big(cp(X, Z) \text{ AND } before(Z, Y)\big) \rightarrow before(X, Y) \tag{14.23}$$

Let us now explore some of the *before* facts that we can prove with the eight Course-Prerequisite facts given at the beginning of the example, plus rules (14.22) and (14.23). First, we can apply rule (14.22) to turn each of the cp facts into a corresponding *before* fact, yielding

$$before(\text{``CS101''}, \text{``CS100''}) \qquad before(\text{``EE200''}, \text{``EE005''})$$
$$before(\text{``EE200''}, \text{``CS100''}) \qquad before(\text{``CS120''}, \text{``CS101''})$$
$$before(\text{``CS121''}, \text{``CS120''}) \qquad before(\text{``CS205''}, \text{``CS101''})$$
$$before(\text{``CS206''}, \text{``CS121''}) \qquad before(\text{``CS206''}, \text{``CS205''})$$

For example, we may use the substitution

$$sub_1(X) = \text{``CS101''}$$
$$sub_1(Y) = \text{``CS100''}$$

on (14.22) to get the substituted rule instance

$$cp(\text{``CS101''}, \text{``CS100''}) \rightarrow before(\text{``CS101''}, \text{``CS100''})$$

This rule, together with the hypothesis $cp(\text{``CS101''}, \text{``CS100''})$, gives us

$$before(\text{``CS101''}, \text{``CS100''})$$

Now we can use rule (14.23) with the hypothesis $cp(\text{``CS120''}, \text{``CS101''})$ and the fact $before(\text{``CS101''}, \text{``CS100''})$ that we just proved, to prove

$$before(\text{``CS120''}, \text{``CS100''})$$

That is, we apply the substitution

$$sub_2(X) = \text{``CS120''}$$
$$sub_2(Y) = \text{``CS100''}$$
$$sub_2(Z) = \text{``CS101''}$$

to (14.23) to obtain the rule

Paths in a Graph

Example 14.24 deals with a common form of rules that define paths in a directed graph, given the arcs of the graph. Think of the courses as nodes, with an arc $a \to b$ if course b is a prerequisite of course a. Then $before(a, b)$ corresponds to the existence of a path of length 1 or more from a to b. Figure 14.9 shows the graph based on the Course-Prerequisite information from Fig. 8.2(b).

When the graph represents prerequisites, we expect it to be acyclic, because it would not do to have a course that had to be taken before itself. However, even if the graph has cycles, the same sort of logical rules define paths in terms of arcs. We can write these rules

$$arc(X, Y) \to path(X, Y)$$

that is, if there is an arc from node X to node Y, then there is a path from X to Y, and

$$\big(arc(X, Z) \text{ AND } path(Z, Y)\big) \to path(X, Y)$$

That is, if there is an arc from X to some Z, and a path from Z to Y, then there is a path from X to Y. Notice that these are the same rules as (14.22) and (14.23), with predicate *arc* in place of *cp*, and *path* in place of *before*.

$$\big(cp(\text{``CS120''}, \text{``CS101''}) \text{ AND } before(\text{``CS101''}, \text{``CS100''})\big)$$
$$\to before(\text{``CS120''}, \text{``CS100''})$$

We then may infer the head of this substituted rule, to prove

$$before(\text{``CS120''}, \text{``CS100''})$$

Similarly, we may apply rule (14.23) to the ground atomic formulas

$$cp(\text{``CS121''}, \text{``CS120''})$$

and $before(\text{``CS120''}, \text{``CS100''})$ to prove $before(\text{``CS121''}, \text{``CS100''})$. Then we use (14.23) on $cp(\text{``CS206''}, \text{``CS121''})$ and $before(\text{``CS121''}, \text{``CS100''})$ to prove

$$before(\text{``CS206''}, \text{``CS100''})$$

There are many other *before* facts we could prove in a similar manner. ❖

EXERCISES

14.9.1*: We can show that the *before* predicate of Example 14.24 is the transitive closure of the *cp* predicate as follows. Suppose there is a sequence of courses $c_1, c_2, \ldots, c_n$, for some $n \geq 2$, and c_1 is a prerequisite of c_2, which is a prerequisite of c_3, and so on; in general, $cp(c_i, c_{i+1})$ is a given fact for $i = 1, 2, \ldots, n-1$. Show that c_1 comes before c_n by showing that $before(c_1, c_i)$ for all $i = 2, 3, \ldots, n$, by induction on i.

14.9.2: Using the rules and facts of Example 14.24, prove the following facts.

a) $before(\text{``CS120''}, \text{``CS100''})$
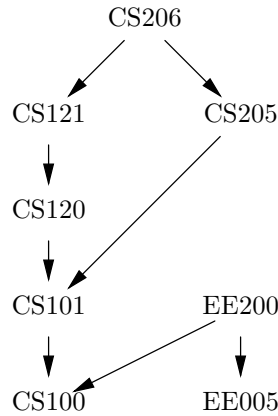b) $before(\text{``CS206''}, \text{``CS100''})$

Fig. 14.9. Prerequisite information as a directed graph.

14.9.3: We can speed up the process of following chains of prerequisites by adding to Example 14.24 the rule

$$\big(before(X, Z)\ \texttt{AND}\ before(Z, Y)\big) \rightarrow before(X, Y)$$

That is, the first subgoal can be any *before* fact, not just a prerequisite fact. Using this rule, find a shorter proof for Exercise 14.9.2(b).

14.9.4*: How many *before* facts can we prove in Example 14.24?

14.9.5: Let *csg* be a predicate that stands for the Course-Student-Grade relation of Fig. 8.1, *cdh* be a predicate that stands for the Course-Day-Hour relation of Fig. 8.2(c), and *cr* a predicate for the Course-Room relation of Fig. 8.2(d). Let *where*(S, D, H, R) be a predicate that means student S is in room R at hour H of day D. More precisely, S is taking a course that meets in that room at that time.

a)   Write a rule that defines *where* in terms of *csg*, *cdh*, and *cr*.

b)   If the facts for the predicates *csg*, *cdh*, and *cr* are as given in Figs. 8.1 and 8.2, what *where* facts can you infer? Give a proof of two such facts.

❖ 14.10   Truth and Provability

We close our discussion of predicate logic with an introduction to one of the more subtle issues of logic: the distinction between what is provable and what is true. We have seen inference rules that allow us to prove things in either propositional or predicate logic, yet we could not be sure that a given set of rules was complete, in the sense that they allowed us to prove every true statement. We asserted, for instance, that resolution as we presented it in Section 12.11 is complete for propositional logic. A generalized form of resolution, which we do not cover here, is also complete for predicate logic.

Models

However, to understand completeness of a proof-strategy, we need to grasp the notion of "truth." To get at "truth," we need to understand the notion of a *model*. Every kind of logic has a notion of models for a collection of expressions. These models are the interpretations that make the expressions true.

❖   **Example 14.25.** In propositional logic, the interpretations are truth assignments. Consider the expressions $E_1 = p$ `AND` q and $E_2 = \bar{p}$ `OR` r. There are eight truth assignments for expressions involving variables p, q, and r, which we may denote by a string of three bits, for the truth values of each of these variables, in order.

   The expression E_1 is made true only by those assignments that make both p and q true, that is, by 110 and 111. The expression E_2 is made true by six assignments: 000, 001, 010, 011, 101, and 111. Thus there is only one model for set of expressions $\{E_1, E_2\}$, namely 111, since only this model appears on both lists. ❖

❖   **Example 14.26.** In predicate logic, interpretations are the structures defined in Section 14.5. Let us consider the expression E

$$(\forall X)(\exists Y)p(X, Y)$$

That is, for every value of X there is at least one value of Y for which $p(X, Y)$ is true.

   An interpretation makes E true if for every element a of the domain D, there is some element b in D — not necessarily the same b for each a — such that the relation that is the interpretation for predicate p has the pair (a, b) as a member. These interpretations are models of E; other interpretations are not. For example, if domain D is the integers and the interpretation for predicate p makes $p(X, Y)$ true if and only if $X < Y$, then we have a model for expression E. However, the interpretation that has domain D equal to the integers and has the interpretation of p in which $p(X, Y)$ is true if and only if $X = Y^2$, is not a model for expression E. ❖

Entailment

We can now state what it means for an expression E to be true, given a collection of expressions $\{E_1, E_2, \ldots, E_n\}$. We say that $\{E_1, E_2, \ldots, E_n\}$ *entails* expression E if every model M for $\{E_1, E_2, \ldots, E_n\}$ is also a model for E. The *double turnstile* operator $\models$ denotes entailment, as

Double turnstile

$$E_1, E_2, \ldots, E_n \models E$$

The intuition we need is that each interpretation is a possible world. When we say $E_1, E_2, \ldots, E_n \models E$, we are saying that E is true in every possible world where the expressions $E_1, E_2, \ldots, E_n$ are true.

   The notion of entailment should be contrasted with the notion of proof. If we have a particular proof system such as resolution in mind, then we can use the *single turnstile* operator $\vdash$ to denote proof in the same way. That is,

Single turnstile

$$E_1, E_2, \ldots, E_n \vdash E$$

means that, for the set of inference rules at hand, there is a proof of E from the hypotheses $E_1, E_2, \ldots, E_n$. Note that $\vdash$ can have different meanings for different proof systems. Also remember that $\models$ and $\vdash$ are not necessarily the same relationship, although we would generally like to have a proof system in which one is true if and only if the other is true.

There is a close connection between tautologies and entailment. In particular, suppose $E_1, E_2, \ldots, E_n \models E$. Then we claim

$$(E_1 \text{ AND } E_2 \text{ AND } \cdots \text{ AND } E_n) \to E \tag{14.24}$$

is a tautology. Consider some interpretation I. If I makes the left-hand side of (14.24) true, then I is a model of $\{E_1, E_2, \ldots, E_n\}$. Since $E_1, E_2, \ldots, E_n \models E$, interpretation I must also make E true. Thus I makes (14.24) true.

The only other possibility is that I makes the left-hand side of (14.24) false. Then, because an implication is always true when its left-hand side is false, we know (14.24) is again true. Thus (14.24) is a tautology.

Conversely, if (14.24) is a tautology, then we can prove $E_1, E_2, \ldots, E_n \models E$. We leave this proof as an exercise.

Notice that our argument does not depend on whether the expressions involved are of propositional or predicate logic, or some other kind of logic that we have not studied. We only need to know that the tautologies are the expressions made true by every "interpretation" and that a model for an expression or set of expressions is an interpretation making the expression(s) true.

Comparing Provability and Entailment

We would like to know that a given proof system allows us to prove everything that is true and nothing that is false. That is, we want the single and double turnstiles to mean the same thing. A proof system is said to be *consistent* if, whenever something can be proved, it is also entailed. That is, $E_1, E_2, \ldots, E_n \vdash E$ implies $E_1, E_2, \ldots, E_n \models E$. For example, we discussed in Section 12.10 why our inference rules for propositional logic were consistent. To be exact, we showed that whenever we started with hypotheses $E_1, E_2, \ldots, E_n$ and wrote a line E in our proof, then $(E_1 \text{ AND } E_2 \text{ AND } \cdots \text{ AND } E_n) \to E$ was a tautology. By what we argued above, that is the same as saying $E_1, E_2, \ldots, E_n \models E$.

Consistent proof system

We would also like our proof system to be complete. Then we could prove everything that was entailed by our hypotheses, even if finding that proof was hard. It turns out that the inference rules we gave in Section 12.10, or the resolution rule in Section 12.11 are both complete proof systems. That is, if $E_1, E_2, \ldots, E_n \models E$, then $E_1, E_2, \ldots, E_n \vdash E$ in either of these proof systems. There are also complete proof systems for predicate logic, although we shall not introduce them.

Gödel's Incompleteness Theorem

One of the most striking results of modern mathematics is often seen as a contradiction to what we have just said about there being complete proof systems for predicate logic. This result actually concerns not predicate logic as we have discussed it, but rather a specialization of this logic that lets us talk about integers and the usual operations on integers. In particular, we have to modify predicate logic to introduce predicates for the arithmetic operations, such as

1. $plus(X, Y, Z)$, which we want to be true when and only when $X + Y = Z$,
2. $times(X, Y, Z)$, true exactly when $X \times Y = Z$, and
3. $less(X, Y)$, true exactly when $X < Y$.

Further, we need to restrict the domain in interpretations so that the values appear to be the nonnegative integers. We can do that in one of two ways. One way is to introduce a set of expressions that we assert are true. By selecting these expressions properly, the domain in any interpretation satisfying the expressions must "look like" the integers, and the special predicates such as *plus* or *less* must act as would the familiar operations with those names.

✦   **Example 14.27.** We can assert expressions such as

$$plus(X, Y, Z) \rightarrow plus(Y, X, Z)$$

which is the commutative law for addition, or

$$\big(less(X, Y) \text{ AND } less(Y, Z)\big) \rightarrow less(X, Z)$$

which is the transitive law for $<$. ✦

Perhaps a simpler way to understand the restriction of predicate logic that Gödel's theorem addresses is to suppose that the logic allows only one model, the model in which the domain is the nonnegative integers, and the special predicates are given relations corresponding to their conventional meaning. For instance, we would let the interpretation for predicate *plus* be

$$\{(a, b, c) \mid a + b = c\}$$

Gödel's theorem states that no matter what consistent proof system one selects, there is some expression E that is true but unprovable! More precisely, if $E_1, E_2, \ldots, E_n$ is a set of expressions all of whose models behave as do the nonnegative integers, then $E_1, E_2, \ldots, E_n \models E$ is true, yet $E_1, E_2, \ldots, E_n \vdash E$ is false. That is, there is no proof of E from $\{E_1, E_2, \ldots, E_n\}$ in our chosen system.

The unprovable expression E may be different for different chosen proof systems. In fact, the selected expression E can be thought of as a way to encode into integers the fact that the expression itself has no proof in the given proof system.

Limits on What a Computer Can Do

An important consequence of Gödel's theorem is that there is a limit on our ability to answer questions about mathematics. If we have a mathematical model as complex as the integers (and many mathematical models are far more complex than integers, as we have seen in this book), then there is no mechanical way we can distinguish true statements from false ones. The best we can do is use some consistent proof system to allow us to search for proofs. If we find one, we are lucky, and we can be sure that the proved statement is true. However, our search may go on forever, without ever finding a proof, even though the statement is true; that is, the statement is entailed by the assumptions we have made to define the mathematical model at hand.

Philosophically, this situation suggests that mathematics will forever remain interesting and challenging. Practically, it suggests that there are limits to what

Undecidability

The logician Alan Turing developed a formal theory of computing in the 1930's considerably before there were any electronic computers to model with his theory. The most important result of this theory is the discovery that certain problems are *undecidable*; no computer whatsoever can answer them.

Turing machine   A centerpiece of the theory is the *Turing machine*, an abstract computer that consists of a finite automaton with an infinite tape divided into squares. In a single move, the Turing machine can read the character on the one square seen by its *tape head*, and based on that character and its current state, replace the character by a different one, change its state, and move the tape head one square left or right. An observed fact is that every real computer, as well as every other mathematical model of what a computing engine should be, can compute exactly what the Turing machine can compute. Thus we take the Turing machine as the standard abstract model of a computer.

However, we do not have to learn the details of what a Turing machine can do in order to appreciate Turing's theory. It suffices to take as a model of a computer a kind of C program that reads character input and has only two possible write statements: `printf("yes\n")` and `printf("no\n")`. Moreover, after making an output of either type, the program must terminate, so that it cannot make a contradictory output later. Understand that a program of this type might, on some inputs, give neither a "yes" nor a "no" response; it might run forever in a loop.
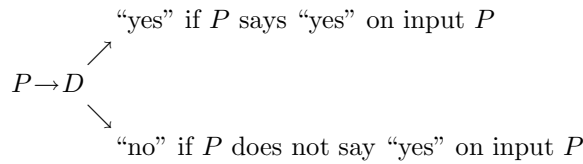
We shall prove that there is no program like D, the "decider" program of Fig. 14.10(a). D supposedly takes as input a program P of the special type above, and says "yes" if P says "yes" when given P itself as input. D says "no" if — when P is given P as input — P either says "no" or P fails to make any decision. As we shall see, it is this requirement that D figure out the occasions when P is never going to render a decision that makes D impossible to write.

However, supposing that D exists, it is a simple matter to write a "complementer" program C, as suggested in Fig. 14.10(b). C is formed from the hypothetical D by changing every statement that prints "no" into one that prints "yes," and vice versa.
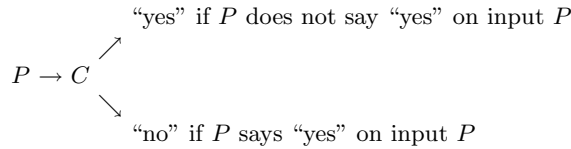
Now we ask what happens when C is given itself as input, as suggested in Fig. 14.10(c)? If C says "yes," then as Fig. 14.10(b) reminds us, C is asserting that "C does not say 'yes' on input C." If C says "no," then C is asserting that "C says 'yes' on input C." We now have a contradiction similar to Russell's paradox, where C can say neither "yes" nor "no" truthfully.

The conclusion is that the decider program D does not really exist. That is, the problem solved by D, which is whether a given C program of the restricted type says "yes" or fails to say "yes" (by saying "no" or by saying nothing) when given itself as input, cannot be solved by computer. It is an undecidable problem.
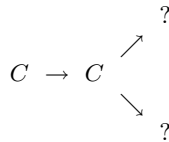
Since Turing's original result, a wide variety of undecidable problems have been discovered. For example, it is undecidable whether a given C program enters an infinite loop on a given input, or whether two C programs produce the same output on the same input.

"yes" if P says "yes" on input P

$P \rightarrow D$

"no" if P does not say "yes" on input P

(a) The hypothetical decider D.

"yes" if P does not say "yes" on input P

$P \rightarrow C$

"no" if P says "yes" on input P

(b) The complementer C.

?

$C \rightarrow C$

?

(c) What does C do with itself as input?

Fig. 14.10. Parts of Turing's construction.

we can do with a computer. In particular, we cannot write programs to find proofs in sufficiently complex systems, although we can do so in simple systems, such as propositional logic or even predicate logic without any special predicates or restrictions. The reader should observe the box on "Undecidability," in which a theorem related to Gödel's is discussed. The theory of undecidability allows us to point to specific problems that we can show no computer can solve.

Thus, rather than ending this book on a negative note, the theory of undecidability reminds us that, like mathematics, computer science is destined to challenge the best minds the human race can produce. The student who pursues the subject will learn the art and science that is needed to avoid the undecidable (and the intractable as well). He or she may then join the ranks of those scientists and engineers pushing back the frontiers of what our computing machines can do.

EXERCISES

14.10.1: Let $E_1 = p$, $E_2 = q$, and $E_3 = qr + p\bar{r}$. Describe the models (truth assignments for propositional variables p, q, and r) that make $\{E_1, E_2\}$ true. Describe the models for E_3. Is $E_1, E_2 \models E_3$ true? Why or why not?

14.10.2**: Consider the following expressions of predicate logic.

1. $E_1 = (\forall X)p(X, X)$

2. $E_2 = (\forall X)(\forall Y)\big(p(X, Y) \rightarrow p(Y, X)\big)$

3. $E_3 = (\forall X)(\forall Y)(\forall Z)\Big(\big(p(X, Y) \text{ AND } p(Y, Z)\big) \rightarrow p(X, Z)\Big)$

4. $E_4 = (\forall X)(\forall Y)\big(p(X, Y) \text{ OR } p(Y, X)\big)$

5. $E_5 = (\forall X)(\exists Y)p(X, Y)$

Which of these five expressions are entailed by the other four? In each case, either give an argument about all possible interpretations, to show entailment, or give a particular interpretation that is a model of four of the expressions but not the fifth. *Hint*: Start by imagining that the predicate p represents the arcs of a directed graph, and look at each expression as a property of graphs. The material in Section 7.10 should give some hints either for finding appropriate models in which the domain is the nodes of a certain graph and predicate p the arcs of that graph, or for showing why there must be entailment. Note however, that it is not sufficient to show entailment by insisting that the interpretation be a graph.

14.10.3*: Let S_1 and S_2 be two sets of expressions of predicate logic (or propositional logic — it doesn't matter), and let their corresponding sets of models be M_1 and M_2, respectively.

a)   Show that the set of models for the set of expressions $S_1 \cup S_2$ is $M_1 \cap M_2$.
b)   Is the set of models for set of expressions $S_1 \cap S_2$ always equal to $M_1 \cup M_2$?

14.10.4*: Show that if $(E_1 \text{ AND } E_2 \text{ AND } \cdots \text{ AND } E_n) \rightarrow E$ is a tautology, then $E_1, E_2, \ldots, E_n \models E$.

❖ 14.11   Summary of Chapter 14

The reader should have learned the following points from this chapter.

❖   Predicate logic uses atomic formulas, that is, predicates with arguments, as atomic operands and the operators of propositional logic, plus the two quantifiers, "for all" and "there exists."

❖   Variables in an expression of predicate logic are bound by quantifiers in a manner analogous to the binding of variables in a program to declarations.

❖   Instead of the truth assignments of propositional logic, in predicate logic we have a complex structure called an "interpretation." An interpretation consists of a domain of values, relations on that domain for the predicates, and values from the domain for any free variables.

❖   The interpretations that make a set of expressions true are the "models" of that set of expressions.

❖   Tautologies of predicate calculus are those that evaluate to TRUE for every interpretation. While many tautologies are obtained by substitution into tautologies of propositional logic, there are also some important tautologies involving quantifiers.

❖ It is possible to put every expression of predicate logic into "prenex form," consisting of a quantifier-free expression to which quantifiers are applied as the last operators.

❖ Proofs in predicate logic can be constructed in a manner similar to proofs in propositional logic.

❖ The substitution of constants for variables in tautologies yields another tautology, and this inference rule is useful in proofs, especially when we work from a database of facts and a collection of rules.

❖ A set of expressions $\{E_1, \ldots, E_n\}$ "entails" an expression E if any model of the former is also a model of the latter. We regard E as "true" given $E_1, \ldots, E_n$ as hypotheses if E is entailed by $E_1, \ldots, E_n$.

❖ Gödel's theorem states that if we take expressions describing number theory (i.e., arithmetic for the nonnegative integers) as hypotheses, then for any proof system, there is some expression that is entailed by the hypotheses but cannot be proved from them.

❖ Turing's theorem describes a formal model of a computer called a "Turing machine" and says that there are problems that cannot be solved by a computer.

❖❖❖ 14.12 Bibliographic Notes for Chapter 14

The books on logic, including Enderton [1972], Mendelson [1987], Lewis and Papadimitriou [1981], and Manna and Waldinger [1990] that we cited in Section 12.14, also cover predicate logic.

Gödel's incompleteness theorem appeared in Gödel [1931]. Turing's paper on undecidability is Turing [1936].

Gödel, K. [1931]. "Uber formal unentscheidbare satze der Principia Mathematica und verwander systeme," *Monatschefte fur Mathematik und Physik* **38**, pp. 173–198.

Turing, A. M. [1936]. "On computable numbers with an application to the *entscheidungsproblem*," *Proc. London Math. Soc.* **2**:42, pp. 230–265.