2

Predicate logic

2.1 The need for a richer language

In the first chapter, we developed propositional logic by examining it from three different angles: its proof theory (the natural deduction calculus), its syntax (the tree-like nature of formulas) and its semantics (what these formulas actually mean). From the outset, this enterprise was guided by the study of declarative sentences, statements about the world which can, in principle, be given a truth value.

We begin this second chapter by pointing out the limitations of propositional logic with respect to encoding purely declarative sentences. Propositional logic dealt quite satisfactorily with sentence components like *not*, *and*, *or* and *if ... then*, but the logical aspects of natural and artificial languages are much richer than that. What can we do with modifiers like *there exists ...*, *all ...*, *among ...* and *only ...*? Here, propositional logic shows clear limitations and the desire to express more subtle declarative sentences led to the design of *predicate logic*, which is also called *first-order logic*.

Let us consider the declarative sentence

Every student is younger than some instructor.

In propositional logic, we could identify this assertion with a propositional atom p. However, that is a rather crude way of reflecting the finer logical structure of this sentence. What is this statement about? Well, it is about *being a student*, *being an instructor* and *being younger than somebody else*. These are all properties of some sort, so we would like to have a mechanism for expressing them together with their logical relationships and dependences. We now use *predicates* for that purpose. For example, we could write

$$S(andy)$$

to denote that Andy is a student and

$$I(paul)$$

to say that Paul is an instructor. Likewise,

$$Y(andy, paul)$$

could mean that Andy is younger than Paul. The symbols S, I and Y are called predicates. Of course, we have to be clear about their meaning. The predicate Y could also mean that the second person is younger than the first one, so we need to specify exactly what these symbols refer to.

Having such predicates at our disposal, we still need to formalise those parts of the sentence above which speak of *every* and *some*. Obviously, this sentence refers to the individuals that make up some academic community (left implicit by the sentence), like Kansas State University or the University of Birmingham, and it says that for each student among them there is an instructor among them such that the student is younger than the instructor.

These predicates are not yet enough to allow us to express the sentence 'Every student is younger than some instructor'. We don't really want to write down all instances of $S(\cdot)$ where $\cdot$ is replaced by every student's name in turn. Similarly, when trying to codify a sentence having to do with the execution of a program, it would be rather laborious to have to write down every state of the computer. Therefore, we employ the concept of a *variable*. Variables are written

$$u, v, w, x, y, z, \ldots \qquad \text{or} \qquad x_1, y_3, u_5, \ldots$$

and can be thought of as *place holders* for concrete values (like a student, or a program state). Using variables, we can now specify the meanings of S, I and Y more formally:

$$\begin{aligned} S(x) &: \quad x \text{ is a student} \\ I(x) &: \quad x \text{ is an instructor} \\ Y(x, y) &: \quad x \text{ is younger than } y. \end{aligned}$$

Note that the names of the variables are not important, provided that we use them consistently. We can state the intended meaning of I by writing

$$I(y) : \quad y \text{ is an instructor}$$

or, equivalently, by writing

$$I(x) : \quad x \text{ is an instructor.}$$

Variables are mere place holders for objects. The availability of variables is still not sufficient for capturing the essence of the example sentence above. We need to convey the meaning of

Every *student* x *is younger than* **some** *instructor* y.

This is where we need to introduce *quantifiers*

$\forall$ (read: 'for all')

$\exists$ (read: 'there exists')

which always come attached to a variable, as in $\forall x$ ('for all x') or in $\exists z$ ('there exists z', or 'there is some z'). Now we can write the example sentence in an entirely symbolic way as

$$\forall x\,(S(x) \rightarrow (\exists y\,(I(y) \wedge Y(x, y)))).$$

Actually, this encoding is rather a paraphrase of the original sentence. In our example, the re-translation results in

For every x, if x is a student, then there is some y which is an instructor such that x is younger than y.

Different predicates can have a different number of arguments. The predicates S and I have just one (they are called *unary predicates*), but predicate Y requires two arguments (it is called a *binary predicate*). Predicates with any finite number of arguments are possible in predicate logic.

Another example is the sentence

Not all birds can fly.

For that we choose the predicates B and F which have one argument expressing

$B(x)$: x is a bird

$F(x)$: x can fly.

The sentence 'Not all birds can fly' can now be coded as

$$\neg(\forall x\,(B(x) \rightarrow F(x)))$$

saying: 'It is not the case that all things which are birds can fly.'. Alternatively, we could code this as

$$\exists x\,(B(x) \wedge \neg F(x))$$

meaning: 'There is some x which is a bird and cannot fly.'. Note that the

first version is closer to the linguistic structure of the sentence above. These two formulas should evaluate to T in the world we currently live in since, for example, penguins are birds which cannot fly. Shortly, we address how such formulas can be given their meaning in general. We will also explain why formulas like the two above are indeed equivalent *semantically.*

Coding up complex facts expressed in English sentences as logical formulas in predicate logic is important and much more care must be taken than in the case of propositional logic. However, once this translation has been accomplished the main objective is to reason symbolically ($\vdash$) or semantically ($\models$) about the information expressed in those formulas.

In Section 2.4, we develop the proper notion of models, real or artificial worlds in which these assertions can be true or false, which allows us to define semantic entailment

$$\phi_1, \phi_2, \ldots, \phi_n \models \psi.$$

The latter expresses that, given *any* such model in which all $\phi_1, \phi_2, \ldots, \phi_n$ hold, it is the case that ψ holds in that model as well. In that case, one also says that ψ is *semantically entailed* by $\phi_1, \phi_2, \ldots, \phi_n$. Although this definition of semantic entailment closely matches the one for propositional logic in Definition 1.33, the process of *evaluating a predicate formula* is quite different from the computation of truth values for propositional logic. We discuss it in detail in Section 2.4.

In Section 2.3, we extend our natural deduction calculus so that it covers logical formulas of predicate logic as well. In this way we are able to prove sequents

$$\phi_1, \phi_2, \ldots, \phi_n \vdash \psi$$

in a similar way to that in the first chapter. It is outside the scope of this book to show that the natural deduction calculus for predicate logic is sound and complete with respect to semantic entailment; but it is indeed the case that

$$\phi_1, \phi_2, \ldots, \phi_n \vdash \psi \quad \text{iff} \quad \phi_1, \phi_2, \ldots, \phi_n \models \psi$$

for formulas of the predicate calculus. The first proof of this was done by the mathematician K. Gödel.

What kind of reasoning must predicate logic be able to support? To get a feel for that, let us consider the following argument:

No books are gaseous. Dictionaries are books. Therefore, no dictionary is gaseous.

The predicates we choose are

$B(x)$: x is a book
$G(x)$: x is gaseous
$D(x)$: x is a dictionary.

Evidently, we need to build a proof theory and semantics that allow us to derive

$$\neg\exists x\,(B(x) \wedge G(x)),\ \forall x\,(D(x) \rightarrow B(x)) \vdash \neg\exists x\,(D(x) \wedge G(x))$$

as well as

$$\neg\exists x\,(B(x) \wedge G(x)),\ \forall x\,(D(x) \rightarrow B(x)) \models \neg\exists x\,(D(x) \wedge G(x)).$$

Verify that this sequent expresses the argument above in a symbolic form.

Predicate symbols and variables allow us to code up much more of the logical structure of declarative sentences than was possible in propositional logic. Predicate logic contains one more concept, that of *function symbols*, that allows us to go even further. Consider the declarative sentence

Every child is younger than its mother.

We could code it using predicates as

$$\forall x\,\forall y\,(C(x) \wedge M(y,x) \rightarrow Y(x,y))$$

where $C(x)$ means that x is a child, $M(x,y)$ means that x is y's mother and $Y(x,y)$ means that x is younger than y. (Note that we actually used $M(y,x)$ (y is x's mother), not $M(x,y)$.) As we have coded it, the sentence says that, for all children x and any mother of theirs y, x is younger than y. It is not very elegant to say 'any of x's mothers', since we know that every individual has one and only one mother[1]. The inelegance of coding 'mother' as a predicate is even more apparent if we consider the sentence

Andy and Paul have the same maternal grandmother.

which in predicate logic, using a and p for Andy and Paul and M for mother as before, becomes

$$\forall x\,\forall y\,\forall u\,\forall v\,(M(x,y) \wedge M(y,a) \wedge M(u,v) \wedge M(v,p) \rightarrow x = u).$$

This formula says that, if y and v are Andy's and Paul's mothers, respectively, and x and u are *their* mothers (i.e. Andy's and Paul's maternal grandmothers, respectively), then x and u are the same person. Notice that we used a special

[1] We assume here that we are talking about genetic mothers, not adopted mothers, step mothers etc.

predicate in predicate logic, *equality*; it is a binary predicate, i.e. it takes two arguments, and is written $=$. Unlike other predicates, it is usually written in between its arguments rather than before them; that is, we write $x = y$ instead of $=(x, y)$ to say that x and y are equal.

The function symbols of predicate logic give us a way of avoiding this ugly encoding, for they allow us to represent y's mother in a more direct way. Instead of writing $M(x, y)$ to mean that x is y's mother, we simply write $m(y)$ to mean y's mother. The symbol m is a function symbol: it takes one argument and returns the mother of that argument. Using m, the two sentences above have simpler encodings than they had using M:

$$\forall x\,(C(x) \rightarrow Y(x, m(x)))$$

now expresses that every child is younger than its mother. Note that we need only one variable rather than two. Representing that Andy and Paul have the same maternal grandmother is even simpler; it is written

$$m(m(a)) = m(m(p))$$

quite directly saying that Andy's maternal grandmother is the same person as Paul's maternal grandmother.

One can always do without function symbols, by using a predicate symbol instead. However, it is usually neater to use function symbols whenever possible, because we get more compact encodings. However, function symbols can be used only in situations in which we want to denote a single object. We rely on the fact that every individual has a uniquely defined mother, so that we can talk about x's mother without risking any ambiguity (for example, if x had no mother, or two mothers). For this reason, we cannot have a function symbol $b(\cdot)$ for 'brother'. It might not make sense to talk about x's brother, for x might not have any brothers, or he might have several. 'Brother' must be coded as a binary predicate.

To exemplify this point further, if Mary has several brothers, then the claim that 'Ann likes Mary's brother' is ambiguous. It might be that Ann likes one of Mary's brothers, which we would write as

$$\exists x\,(B(x, m) \wedge L(a, x))$$

(where B and L mean 'is brother of' and 'likes', and a and m mean Ann and Mary) — this sentence says that there exists an x which is a brother of Mary and is liked by Ann. Alternatively, if Ann likes all of Mary's brothers, we write it as

$$\forall x\,(B(x, m) \rightarrow L(a, x))$$

saying that any x which is a brother of Mary is liked by Ann.

Different function symbols may take different numbers of arguments. In a domain involving students and the grades they get in different courses, one might have the binary function symbol $g(\cdot,\cdot)$ taking two arguments: $g(x,y)$ refers to the grade obtained by student x in course y.

2.2 Predicate logic as a formal language

The discussion of the preceding section was intended to give an impression of how we code up sentences as formulas of predicate logic. In this section, we will be more precise about it, giving syntactic rules for the formation of predicate logic formulas. Because of the power of predicate logic, the language is much more complex than that of propositional logic.

The first thing to note is that there are two *sorts* of things involved in a predicate logic formula. The first sort denotes the objects that we are talking about: individuals such as a and p (referring to Andy and Paul) are examples, as are variables such as x and v. Function symbols also allow us to refer to objects: thus, $m(a)$ and $g(x,y)$ are also objects. Expressions in predicate logic which denote objects are called *terms*.

The other sort of things in predicate logic denotes truth values; expressions of this kind are *formulas*. $Y(x,m(x))$ is a formula, though x and $m(x)$ are terms.

A predicate vocabulary consists of three sets: a set of predicate symbols $\mathscr{P}$, a set of function symbols $\mathscr{F}$ and a set of constant symbols $\mathscr{C}$. Each predicate symbol and each function symbol comes with an arity, the number of arguments it expects.

2.2.1 Terms

The terms of our language are made up of variables, constant symbols and functions applied to those. Functions may be nested, as in $m(m(x))$ or $g(m(a),c)$: the grade obtained by Andy's mother in the course c.

Definition 2.1 *Terms* are defined as follows.

- Any variable is a term.
- Any constant in $\mathscr{C}$ is a term.
- If $t_1, t_2, \ldots, t_n$ are terms and $f \in \mathscr{F}$ has arity n, then $f(t_1, t_2, \ldots, t_n)$ is a term.
- Nothing else is a term.

In Backus Naur form we may write

$$t ::= x \mid c \mid f(t, \ldots, t)$$

where x is a variable, $c \in \mathscr{C}$ and $f \in \mathscr{F}$ has arity n.

It is important to note that

- The first building blocks of terms are *constants* and *variables.*
- More complex terms are built from function symbols using as many previously built terms as arguments as the function symbol in question requires.
- The notion of terms is dependent on the sets $\mathscr{C}$ and $\mathscr{F}$. If you change those, you change the set of terms.

We said that a predicate vocabulary is given by three sets, $\mathscr{P}$, $\mathscr{F}$ and $\mathscr{C}$. In fact, constants can be thought of as functions which don't take any arguments (and we even drop the argument brackets) — therefore, constants live in the set $\mathscr{F}$ together with the 'true' functions which do take arguments. From now on, we will drop the set $\mathscr{C}$, since it is convenient to do so, and stipulate that constants are 0-arity functions.

See Figure 2.1 for the parse tree of the term $(2 - (s(x) + y)) * x$, where $+$, $-$ and $*$ are written in infix.

EXERCISES 2.1

1. Let $\mathscr{F}$ be $\{d, f, g\}$, where d is a constant, f a function symbol with two arguments and g a function symbol with three arguments. Which of the following strings are terms over $\mathscr{F}$? Draw the parse tree of those strings which are indeed terms.

   (a) $g(d, d)$
   * (b) $f(x, g(y, z), d)$
   * (c) $g(x, f(y, z), d)$
   (d) $g(x, h(y, z), d)$
   (e) $f(f(g(d, x), f(g(d, x), y, g(y, d)), g(d, d)), g(f(d, d, x), d), z)$.

2. Let $\mathscr{F}$ be as in the last exercise.

   (a) The length of a term over $\mathscr{F}$ is the length of its string representation, where we count all commas and parentheses. For example, the length of $f(x, g(y, z), z)$ is 13. Can you list all terms over $\mathscr{F}$ which do not contain any variables and whose length is less than 10?

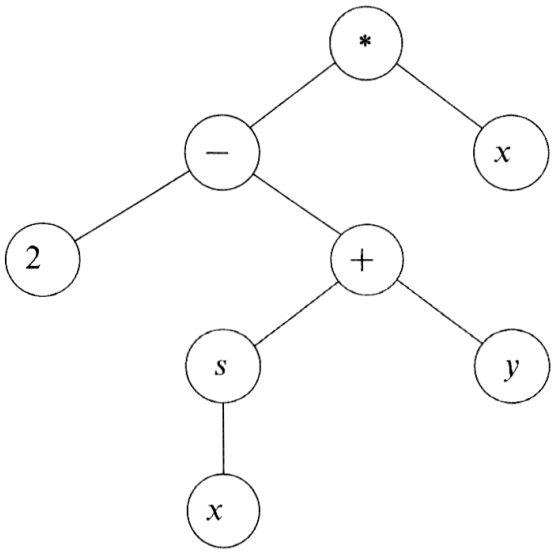

Fig. 2.1. A parse tree representing an arithmetic term.

* (b) The height of a term over $\mathscr{F}$ is defined as 1 plus the length of the longest path in its parse tree, as in Definition 1.31. List all terms over $\mathscr{F}$ which do not contain any variables and whose height is less than 4.

3. Let $\mathscr{F}$ be the set $\{+, -, *, s\}$ where $+, -, *$ are binary functions and s is a unary function. Let $\mathscr{C}$ be the set $\{0, 1, 2, \ldots\}$. We write $+, -, *$ in infix notation rather than prefix notation (that is, we write $x + y$ instead of $+(x, y)$, etc.). Figure 2.1 shows the parse tree of the term $(2-(s(x)+y))*x$. Draw the parse tree of the term $(2-s(x))+(y*x)$. Compare your solution with the parse tree in Figure 2.1.

2.2.2 Formulas

Suppose that our predicate vocabulary is given by the sets of function symbols $\mathscr{F}$ and predicate symbols $\mathscr{P}$. The choice of predicate, function and constant symbols is driven by what we intend to describe. For example, if we work on a database representing relations between our kin we might want to consider

$$\{M, F, S, D\}$$

referring to *being male*, *being female*, *being a son of* ... and *being a daughter of* .... Naturally, F and M are unary predicates (they take one argument) whereas D and S are binary (taking two).

We already know what the terms over $\mathscr{F}$ are. Given that knowledge, we can now proceed to define the formulas of predicate logic.

Definition 2.2 We define the *set of formulas over* $(\mathscr{F}, \mathscr{P})$ inductively, using the already defined set of terms over $\mathscr{F}$:

- If P is a predicate taking n arguments, $n \geq 1$, and if $t_1, t_2, \ldots, t_n$ are terms over $\mathscr{F}$, then $P(t_1, t_2, \ldots, t_n)$ is a formula.
- If ϕ is a formula, then so is $(\neg\phi)$.
- If ϕ and ψ are formulas, then so are $(\phi \wedge \psi)$, $(\phi \vee \psi)$ and $(\phi \rightarrow \psi)$.
- If ϕ is a formula and x is a variable, then $(\forall x\, \phi)$ and $(\exists x\, \phi)$ are formulas.
- Nothing else is a formula.

Note how the arguments given to predicates are always terms. Let us stress again that the notion of 'formula' depends on the particular choice of constant, function and predicate symbols. We can condense this definition using Backus Naur form (BNF):

$$\phi ::= P(t_1, t_2, \ldots, t_n) \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid (\forall x\, \phi) \mid (\exists x\, \phi) \tag{2.1}$$

where P is a predicate of arity n, t_i are terms and x is a variable. Recall that each occurrence of ϕ on the right-hand side of the $::=$ stands for any formula.

Convention 2.3 For convenience, we retain the usual binding priorities agreed upon in Convention 1.3 and add that $\forall y$ and $\exists y$ bind like $\neg$. Thus, the order is:

- $\neg$, $\forall y$ and $\exists y$ bind most tightly;
- then $\vee$ and $\wedge$;
- then $\rightarrow$.

We also often omit brackets around quantifiers, provided that doing so introduces no ambiguities.

Predicate logic formulas can be represented by parse trees. For example, Figure 2.2 represents the formula $\forall x\,((P(x) \rightarrow Q(x)) \wedge S(x, y))$.

Example 2.4 Consider translating the sentence

Every son of my father is my brother.

into predicate logic. We use a constant m to represent 'me' (or 'I'). This example illustrates that coding facts about real life in predicate logic can be done in a variety of ways. As before, the design choice is whether we represent 'father' as a predicate or as a function symbol.

1. As a predicate. We choose a constant m for 'me', so m is a term, and we choose further $\{S, F, B\}$ as the set of predicates with meanings

$$\begin{aligned} S(x,y) &: \quad x \text{ is a son of } y \\ F(x,y) &: \quad x \text{ is the father of } y \\ B(x,y) &: \quad x \text{ is a brother of } y. \end{aligned}$$

   Then the symbolic encoding of the sentence above is

$$\forall x \, \forall y \, (F(x,m) \wedge S(y,x) \rightarrow B(y,m))$$

   saying: 'For all x and all y, if x is a father of m and if y is a son of x, then y is a brother of m.'.
2. As a function. We keep m, S and B as above and write f for the function which, given an argument, returns the corresponding father. Note that this works only because fathers are *unique*, so f really is a function as opposed to a mere relation.

   The symbolic encoding of the sentence above is now

$$\forall x \, (S(x, f(m)) \rightarrow B(x,m))$$

   meaning: 'For all x, if x is a son of the father of m, then x is a brother of m.'. This statement is much less complex insofar as it involves only one quantifier.

EXERCISES 2.2

* 1. Let m be a constant, f a function symbol with one argument and S and B two predicate symbols, each with two arguments. Which of the following strings are formulas in predicate logic? Specify a reason for failure for strings which aren't.

   (a) $S(m,x)$
   (b) $B(m, f(m))$
   (c) $f(m)$
   (d) $B(B(m,x),y)$
   (e) $S(B(m),z)$
   (f) $(B(x,y) \rightarrow (\exists z \, S(z,y)))$
   (g) $(S(x,y) \rightarrow S(y, f(f(x))))$
   (h) $(B(x) \rightarrow B(B(x)))$.

* 2. Use the predicates

$$\begin{aligned} A(x,y): &\quad x \text{ admires } y \\ B(x,y): &\quad x \text{ attended } y \\ P(x): &\quad x \text{ is a professor} \\ S(x): &\quad x \text{ is a student} \\ L(x): &\quad x \text{ is a lecture} \end{aligned}$$

and the function symbol (= constant)

$$m: \quad \text{Mary}$$

to translate the following into predicate logic:

(a) Mary admires every professor.
(The answer is *not* $\forall x\, A(m, P(x))$; see exercise 1.)
(b) Some professor admires Mary.
(c) Mary admires herself.
(d) No student attended every lecture.
(e) No lecture was attended by every student.
(f) No lecture was attended by any student.

3. Let c and d be constants, f a function symbol with one argument, g a function symbol with two arguments and h a function symbol with three arguments. Further, P and Q are predicate symbols with three arguments. Which of the following strings are formulas in predicate logic? Specify a reason for failure for strings which aren't. Draw parse trees of all strings which are formulas of predicate logic.

(a) $\forall x\, P(f(d), h(g(c,x), d, y))$
(b) $\forall x\, P(f(d), h(P(x,y), d, y))$
(c) $\forall x\, Q(g(h(x, f(d), x), g(x,x)), h(x,x,x), c)$
(d) $\exists z\, (Q(z,z,z) \rightarrow P(z))$
(e) $\forall x \forall y\, (g(x,y) \rightarrow P(x,y,x))$
(f) $Q(c,d,c)$.

4. Use the predicate specifications

$$\begin{aligned} B(x,y): &\quad x \text{ beats } y \\ F(x): &\quad x \text{ is an (American) football team} \\ Q(x,y): &\quad x \text{ is quarterback of } y \\ L(x,y): &\quad x \text{ loses to } y \end{aligned}$$

and the constant symbols

$$\begin{aligned} c: &\quad \text{Wildcats} \\ j: &\quad \text{Jayhawks} \end{aligned}$$

to translate the following into predicate logic.

(a) Every football team has a quarterback.
(b) If the Jayhawks beat the Wildcats, then the Jayhawks do not lose to every football team.
(c) The Wildcats beat some team, which beat the Jayhawks.

* 5. Find appropriate predicates and their specification to translate the following into predicate logic:

(a) All red things are in the box.
(b) Only red things are in the box.
(c) No animal is both a cat and a dog.
(d) Every prize was won by a boy.
(e) A boy won every prize.

6. Let $F(x, y)$ mean that x is the father of y; $M(x, y)$ denotes x is the mother of y. Similarly, $H(x, y)$, $S(x, y)$, and $B(x, y)$ say that x is the husband/sister/brother of y, respectively. You may also use constants to denote individuals, like 'Ed' and 'Patsy'. However, you are not allowed to use any predicate symbols other than the above to translate the following sentences into predicate logic:

(a) Everybody has a mother.
(b) Everybody has a father and a mother.
(c) Whoever has a mother has a father.
(d) Ed is a grandfather.
(e) All fathers are parents.
(f) All husbands are spouses.
(g) No uncle is an aunt.
(h) All brothers are siblings.
(i) Nobody's grandmother is anybody's father.
(j) Ed and Patsy are husband and wife.
(k) Carl is Monique's brother-in-law.

7. Formalise the following sentences in predicate logic, defining predicate symbols as appropriate:

(a) Everybody who visits New Orleans falls in love with it.
(b) There is a trumpet player who lives in New Orleans, but who does not like crawfish étouffée.
(c) There are at least two saxophone players who were born in New Orleans and who play better than every sax player in New York city.

(d) At least two piano players from Louisiana other than Ellis Marsalis play every week at my favourite club.
(e) If the Superdome is as least as high as the Royal Albert Hall, then every concert hall which is as least as high as the Superdome is as least as high as the Royal Albert Hall.
(f) If you eat a po-boy sandwich which has no chicken, no beef, and no seafood in it, then you are eating alligator nuggets.
(g) Abita Amber is the best beer which is brewed in Louisiana.
(h) Mardi Gras is the biggest party in the world.
(i) Not everybody in Louisiana speaks French, but everybody in Louisiana knows someone from Louisiana who does speak French.
(j) Commander's Palace is not only the best restaurant in New Orleans, but also the best one in the United States of America; however, there are restaurants in France which are even better.
(k) There is only one restaurant where you can get better breakfast than at the Bluebird Café.
(l) If you eat red beans and rice for lunch, then it must be a Monday.
(m) Vaughn's is the coolest bar with the best live jazz in New Orleans.
(n) Everybody who talks about the Crescent City actually refers to New Orleans.
(o) Politics in New Orleans is as least as corrupt as that of all Caribbean islands.
(p) Not every hurricane in New Orleans is a storm; some of them are cocktails, but all of them are dangerous.

2.2.3 Free and bound variables

The introduction of variables and quantifiers allows us to express the notions of *all* ... and *some* ... Intuitively, to verify that $\forall x\, Q(x)$ is true amounts to replacing x by any of its possible values and checking that Q holds for each one of them. There are two important and different senses in which such formulas can be 'true'. First, if we fix a certain meaning of all predicate and function symbols involved, then we can *check* whether a formula is true for this particular scenario. For example, if a formula encodes the specifications of a hardware circuit, then we would want to know whether it is true for the

model of the circuit. Second, one sometimes would like to ensure that certain formulas are true *for all models*. Consider $(\forall x\, P(x)) \to (\exists x\, P(x))$; clearly, this formula should be true no matter what model we are looking at. It is this second kind of truth which is the primary focus of this chapter.

Unfortunately, things are more complicated if we want to define formally what it means for a formula to be true in a given model. Ideally, we seek a definition that we could use to write a computer program verifying that a formula holds in a given situation.

To begin with, we need to understand that variables occur in different ways. Consider the formula

$$\forall x\,((P(x) \to Q(x)) \wedge S(x, y)).$$

We draw its parse tree in the same way as for propositional formulas, but with two additional sorts of nodes:

- The quantifiers $\forall x$ and $\exists y$ form nodes and have, like negation, just one subtree.
- Predicates, which are generally of the form $P(t_1, t_2, \ldots, t_n)$, have the symbol P as a node, but now P has n many subtrees, namely the parse trees of the terms $t_1, t_2, \ldots, t_n$.

So in our particular case above we arrive at the parse tree in Figure 2.2. You can see that variables occur at two different sorts of places. First, they appear next to quantifiers $\forall$ and $\exists$ in nodes like $\forall x$ and $\exists z$; such nodes always have one subtree, subsuming their scope to which the respective quantifier applies.

The other sort of occurrence of variables is *leaf nodes containing variables*. If variables are leaf nodes, then they stand for values that still have to be made concrete. There are two principal such occurrences:

1. In our example in Figure 2.2, we have three leaf nodes x. If we walk up the tree beginning at any one of these x leaves, we run into the quantifier $\forall x$. This means that those occurrences of x are actually *bound* to $\forall x$ so they represent, or stand for, *any possible value of* x.
2. In walking upwards, the only quantifier that the leaf node y runs into is $\forall x$ but that x has nothing to do with y; x and y are different place holders. So y is *free* in this formula. This means that its value has to be specified by some additional information, for example, a location in memory.

Definition 2.5 Let ϕ be a formula in predicate logic. An occurrence of x in ϕ is *free* in ϕ if it is a leaf node in the parse tree of ϕ such that there

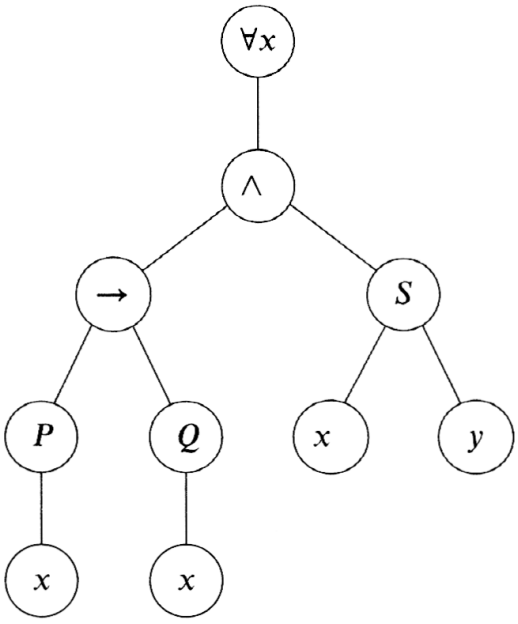

Fig. 2.2. A parse tree of a predicate logic formula.

is no path upwards from that node x to a node $\forall x$ or $\exists x$. Otherwise, that occurrence of x is called *bound*. For $\forall x\,\phi$, or $\exists x\,\phi$, we say that ϕ — minus any of its subformulas $\exists x\,\psi$, or $\forall x\,\psi$ — is the *scope* of $\forall x$, respectively $\exists x$.

Thus, if x occurs in ϕ, then it is bound if, and only if, it is in the scope of $\exists x$ or $\forall x$; otherwise it is free. In terms of parse trees, the scope of a quantifier is just its subtree, minus any subtrees which re-introduce a quantifier for x; e.g. the scope of $\forall x$ in $\forall x\,(P(x) \rightarrow \exists x\,Q(x))$ is $P(x)$. It is quite possible, and common, that a variable is bound and free in a formula. Consider the formula

$$(\forall x\,(P(x) \wedge Q(x))) \rightarrow (\neg P(x) \vee Q(y))$$

and its parse tree in Figure 2.3. The two x leaves in the subtree of $\forall x$ are bound since they are in the scope of $\forall x$, but the leaf x in the right subtree of $\rightarrow$ is free since it is *not* in the scope of any quantifier $\forall x$ or $\exists x$. Note, however, that a single leaf either is under the scope of a quantifier, or it isn't. Hence *individual* occurrences of variables are either free or bound, never both at the same time.

2.2.4 Substitution

Variables are place holders so we must have some means of *replacing* them with more concrete information. On the syntactic side, we often need to

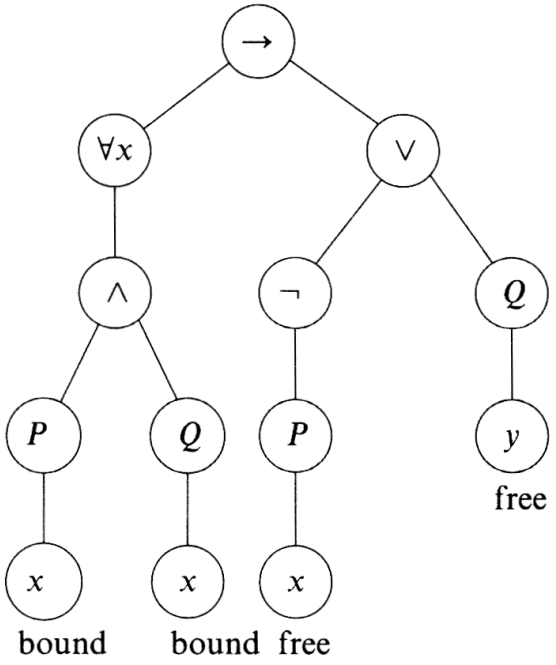

Fig. 2.3. A parse tree of a predicate logic formula illustrating *free* and *bound* occurrences of variables.

replace a leaf node x by the parse tree of an entire term t. Recall from the definition of formulas that any replacement of x may only be a term; it could not be a predicate, or a more complex formula, for x serves as an argument to a predicate one step higher up in the parse tree (see Definition 2.1 and the grammar in (2.1)). In substituting t for x we have to leave untouched the *bound* leaves x since they are in the scope of some $\exists x$ or $\forall x$, i.e. they stand for *some unspecified* or *all* values respectively.

Definition 2.6 Given a variable x, a term t and a formula ϕ we define $\phi[t/x]$ to be the formula obtained by replacing each *free* occurrence of variable x in ϕ with t.

Substitutions are easily understood by looking at some examples. Let f be a function symbol with two arguments and ϕ the formula with the parse tree in Figure 2.2. Then $f(x, y)$ is a term and $\phi[f(x, y)/x]$ is just ϕ again. This is true because *all* occurrences of x are bound in ϕ, so *none* of them gets substituted.

Now consider ϕ to be the formula with the parse tree in Figure 2.3. Here we have one free occurrence of x in ϕ, so we substitute the parse tree of $f(x, y)$ for that free leaf node x and obtain the parse tree in Figure 2.4. Note that the bound x leaves are unaffected by this operation. You can see that

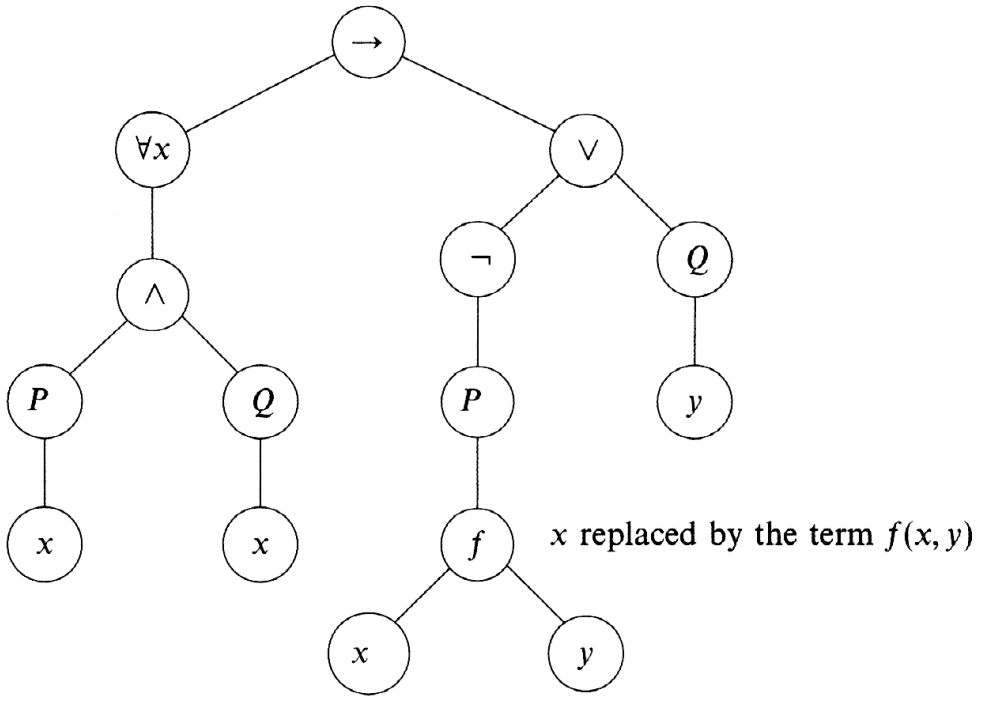

Fig. 2.4. A parse tree of a formula resulting from substitution.

the process of substitution is straightforward, but requires that it be applied *only to the free occurrences* of the variable to be substituted.

A word on notation: in writing $\phi[t/x]$, we really mean this to be the formula *obtained* by performing the operation $[t/x]$ on ϕ. Strictly speaking, the chain of symbols $\phi[t/x]$ is *not* a logical formula, but its *result* will be a formula, provided that ϕ was one in the first place.

Unfortunately, substitutions can give rise to undesired side effects. In performing a substitution $\phi[t/x]$, the term t may contain a variable y, where free occurrences of x in ϕ are under the scope of $\exists y$ or $\forall y$ in ϕ. By carrying out this substitution $\phi[t/x]$, the value y, which might have been fixed by a concrete context, gets caught in the scope of $\exists y$ or $\forall y$. This binding capture overrides the context specification of the concrete value of y, for it will now stand for '*some unspecified*' or '*all*', respectively. Such undesired variable captures are to be avoided at all costs.

Definition 2.7 Given a term t, a variable x and a formula ϕ, we say that t *is free for* x in ϕ if no free x leaf in ϕ occurs in the scope of $\forall y$ or $\exists y$ for any variable y occurring in t.

This definition is maybe hard to swallow. Let us think of it in terms of parse trees. Given the parse tree of ϕ and the parse tree of t, we can perform the

substitution $[t/x]$ on ϕ to obtain the formula $\phi[t/x]$. The latter has a parse tree where all free x leaves of the parse tree of ϕ are replaced by the parse tree of t. What 't is free for x in ϕ' means is that the variable leaves of the parse tree of t won't become bound if placed into the bigger parse tree of $\phi[t/x]$. For example, if we consider x, t and ϕ in Figure 2.4, then t is free in x for ϕ since the *new* leaf variables x and y of t are not under the scope of any quantifiers involving x or y.

As an example where t is not free for x in ϕ, consider the ϕ with parse tree in Figure 2.5 and let t be $f(y, y)$. Then we may substitute the leftmost x leaf since it is not in the scope of any quantifier, but, in substituting the x leaf in the left subtree of $\rightarrow$, we introduce a new variable y in t which becomes bound by $\forall y$.

What if there are no free occurrences of x in ϕ? Inspecting the definition of 't is free for x in ϕ', we see that *every* term t is free for x in ϕ in that case, since no free variable x of ϕ is below some quantifier in the parse tree of ϕ. So the problematic situation of variable capture in performing $\phi[t/x]$ cannot occur. Of course, in that case $\phi[t/x]$ is just ϕ again.

It might be helpful to compare 't is free for x in ϕ' with a precondition of calling a procedure for substitution. If you are asked to compute $\phi[t/x]$ in your exercises or exams, then that is what you should do; but any reasonable implementation of substitution used in a theorem prover would have to check whether t is free for x in ϕ and, if not, rename some variables with fresh ones to avoid the undesirable capture of variables.

EXERCISES 2.3

1. Let ϕ be

$$\exists x\,(P(y,z) \wedge (\forall y\,(\neg Q(y,x) \vee P(y,z)))),$$

where P and Q are predicates with two arguments.

* (a) Draw the parse tree of ϕ.

* (b) Identify those variable leaves which occur free and those which occur bound in ϕ.

(c) Is there a variable in ϕ which has free *and* bound occurrences?

* (d) Consider the terms w (w is a variable), $f(x)$ and $g(y,z)$, where f and g are function symbols with one, respectively two, arguments.

(i) Compute $\phi[w/x]$, $\phi[w/y]$, $\phi[f(x)/y]$ and $\phi[g(y,z)/z]$.

(ii) Which of w, $f(x)$ and $g(y,z)$ are free for x in ϕ?

(iii) Which of w, $f(x)$ and $g(y,z)$ are free for y in ϕ?

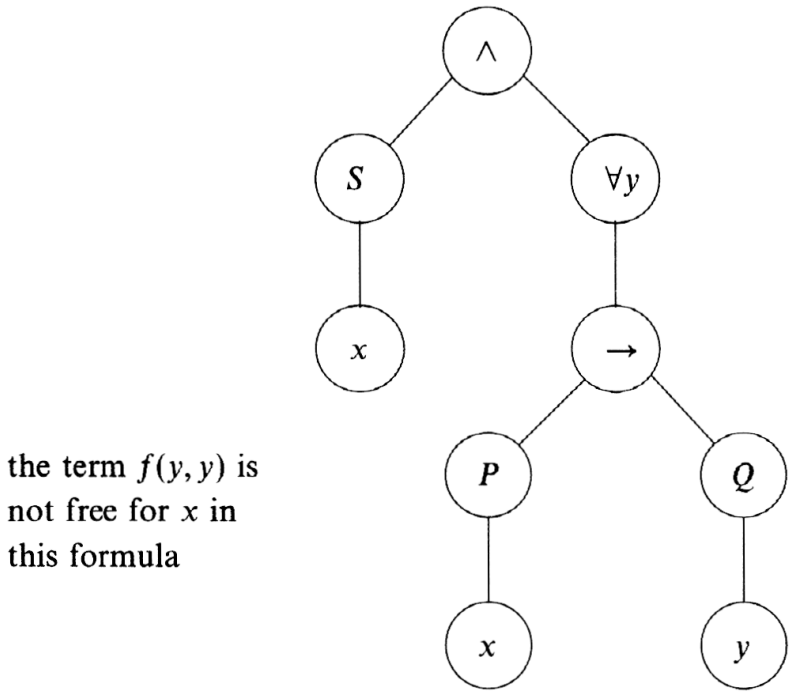

Fig. 2.5. A parse tree for which a substitution has dire consequences.

(e) What is the scope of $\exists x$ in ϕ?

* (f) Suppose that we change ϕ to $\exists x(P(y,z) \wedge (\forall x(\neg Q(x,x) \vee P(x,z))))$. What is the scope of $\exists x$ now?

2. (a) Draw the parse tree of the following logical formula ψ:

$$\neg(\forall x((\exists y\, P(x,y,z)) \wedge (\forall z\, P(x,y,z))))$$

where P is a predicate with three arguments.

(b) Indicate the free and bound variables in that parse tree.

(c) List all variables which occur free *and* bound therein.

(d) Compute $\psi[t/x]$, $\psi[t/y]$ and $\psi[t/z]$, where t equals the term $g(f(g(y,y)),y)$. Is t free for x in ψ? Is t free for y in ψ? Is t free for z in ψ?

2.3 Proof theory of predicate logic

2.3.1 Natural deduction rules

Proofs in the natural deduction calculus for predicate logic are similar to those for propositional logic in Chapter 1, except that we have new rules for dealing with the quantifiers and with the equality symbol. Strictly speaking,

we are *overloading* the previously established rules for the propositional connectives $\wedge$, $\vee$ etc. That simply means that any proof rule of Chapter 1 is still valid for logical formulas of predicate logic (we originally defined those rules for logical formulas of propositional logic). As in the natural deduction calculus for propositional logic, the additional rules for the quantifiers and equality will come in two flavours: introduction and elimination rules.

The proof rules for equality

First, let us state the rules for equality. Here equality does not mean syntactic, or intensional, equality, but equality in terms of computation results. In either of these senses, any term t has to be equal to itself. This is expressed by the introduction rule for equality:

$$\frac{}{t = t}\ =\text{i}$$

which is an axiom (as it does not depend on any premises). Notice that it may be invoked only if t is a term (our language doesn't permit us to talk about equality between formulas).

This rule is quite evidently sound, but it is not very useful on its own. What we need is a principle that allows us to substitute equals for equals repeatedly. For example, suppose that $y * (w + 2)$ equals $y * w + y * 2$; then it certainly must be the case that $z \geq y * (w + 2)$ implies $z \geq y * w + y * 2$ and vice versa. We may now express this substitution principle as the rule =e:

$$\frac{t_1 = t_2 \quad \phi[t_1/x]}{\phi[t_2/x]}\ =\text{e.}$$

Note that t_1 and t_2 have to be free for x in ϕ, whenever we want to apply the rule =e (this is an example of a *side condition* of a proof rule).

Convention 2.8 Indeed, throughout this section, when we write a substitution in the form $\phi[t/x]$, we implicitly assume that t is free for x in ϕ; for, as we saw in the last section, a substitution doesn't make sense if this is not the case.

We obtain proof

1	$(x + 1) = (1 + x)$	premise
2	$(x + 1 > 1) \rightarrow (x + 1 > 0)$	premise
3	$(1 + x > 1) \rightarrow (1 + x > 0)$	=e 1, 2

establishing the validity of the sequent

$$x + 1 = 1 + x, (x + 1 > 1) \rightarrow (x + 1 > 0) \vdash (1 + x) > 1 \rightarrow (1 + x) > 0.$$

In this particular proof t_1 is $(x+1)$, t_2 is $(1+x)$ and ϕ is $(x > 1) \to (x > 0)$. We used the name =e since it reflects what this rule is doing to data: it eliminates the equality in $t_1 = t_2$ by replacing all t_1 in $\phi[t_1/x]$ with t_2. This is a sound substitution principle, since the assumption that t_1 equals t_2 guarantees that the logical meanings of $\phi[t_1/x]$ and $\phi[t_2/x]$ match.

The principle of substitution, in the guise of the rule =e, is quite powerful. Together with the rule =i, it allows us to show the sequents

$$t_1 = t_2 \vdash t_2 = t_1$$
$$t_1 = t_2, t_2 = t_3 \vdash t_1 = t_3.$$

A proof for the first sequent is:

1	$t_1 = t_2$	premise
2	$t_1 = t_1$	=i
3	$t_2 = t_1$	=e 1,2

where ϕ is $x = t_1$.

A proof for the second sequent is:

1	$t_2 = t_3$	premise
2	$t_1 = t_2$	premise
3	$t_1 = t_3$	=e 1,2

where ϕ is $t_1 = x$, so in line 2 we have $\phi[t_2/x]$ and in line 3 we obtain $\phi[t_3/x]$, as given by the rule =e applied to lines 1 and 2. Notice how we applied the scheme =e with several different instantiations.

Our discussion of the rules =i and =e has shown that they force equality to be *reflexive*, *symmetric* and *transitive*. These are minimal and necessary requirements for any sane concept of (extensional) equality. We leave the topic of equality for now to move on to the proof rules for quantifiers.

EXERCISES 2.4

1. Prove the following sequents using, among others, the rules =i and =e. Make sure that you indicate for each application of =e what the rule instances ϕ, t_1 and t_2 are.
   (a) $(y = 0) \wedge (y = x) \vdash 0 = x$
   (b) $t_1 = t_2 \vdash (t + t_2) = (t + t_1)$
   (c) $(x = 0) \vee ((x+x) > 0) \vdash (y = (x+x)) \to ((y > 0) \vee (y = (0+x)))$.

2. Recall that we use $=$ to express the equality of elements in our models. Consider the formula

$$\exists x \exists y (\neg(x = y) \wedge (\forall z ((z = x) \vee (z = y)))).$$

Although we have not yet formally defined what a model $\mathcal{M}$ for predicate logic looks like, can you say intuitively what this formula says about any such model $\mathcal{M}$ in plain English?

* 3. Write down a sentence ϕ_3 of predicate logic which intuitively holds in a model $\mathcal{M}$ if, and only if, that model has exactly three concrete values.

4. Write down a sentence $\phi_{\leq 3}$ of predicate logic which intuitively holds in a model $\mathcal{M}$ iff that model has at most three concrete values.

* 5. Can you find a sentence of predicate logic $\phi_{<\infty}$ which intuitively holds exactly in those models which have only finitely many concrete values? What 'limitation' of predicate logic causes problems in finding such a sentence?

The proof rules for universal quantification

The rule for eliminating $\forall$ is the following:

$$\frac{\forall x \phi}{\phi[t/x]} \forall x \text{e.}$$

It says: if you have $\forall x \phi$, then you could replace the x in ϕ by any term t (given, as usual, the side condition that t be free for x in ϕ). The intuitive soundness of this rule is evident: assuming that $\forall x \phi$ holds, we should certainly be entitled to maintain that $\phi[t/x]$ holds, where t is some term.

Recall that $\phi[t/x]$ is obtained by replacing all free occurrences of x in ϕ by t. You may think of the term t as a concrete *instance* of x. Since ϕ is assumed to hold for all x, that should also be the case for any term t. To see the necessity of the proviso that t be free for x in ϕ, consider the case that ϕ is

$$\exists y (x < y)$$

and the term to be substituted for x is y. Let's suppose we are reasoning about numbers with the usual 'smaller than' relation. The statement $\forall x \phi$ then says that for all numbers n there is some bigger number m, which is indeed true of integers or real numbers. However, $\phi[y/x]$ is the formula

$$\exists y (y < y)$$

saying that there is a number which is bigger than itself. This is wrong; and we must not allow a proof rule which derives semantically wrong things from semantically valid ones. Clearly, what went wrong was that y became bound in the process of substitution; y is not free for x in ϕ. Thus, in going from $\forall x\,\phi$ to $\phi[t/x]$, we have to enforce the side condition that t be free for x in ϕ.

The rule $\forall x\,\mathrm{i}$ is a bit more complicated. It employs a proof box similar to those we have already seen in natural deduction for propositional logic, but this time the box is to stipulate the scope of the 'dummy variable' x_0 rather than the scope of an assumption. The rule $\forall x\,\mathrm{i}$ is written

$$\dfrac{\boxed{\begin{array}{l} x_0 \\ \quad\vdots \\ \quad \phi[x_0/x] \end{array}}}{\forall x\,\phi}\ \forall x\,\mathrm{i}$$

It says: if, starting with a 'fresh' variable x_0, you are able to prove some formula with x_0 in it, then (*because* x_0 *is fresh*) you can derive $\forall x\,\phi$. The important point is that x_0 is a new variable which doesn't occur *anywhere outside the box*; we think of it as an *arbitrary* term. Since we assumed nothing about this x_0, anything would work in its place; hence the conclusion $\forall x\,\phi$.

It takes a while to understand this rule, since it seems to be going from the particular case of ϕ to the general case $\forall x\,\phi$. The side condition, that x_0 does not occur outside the box, is what allows us to get away with this. In particular, the formula $\phi[x_0/x]$ may only depend on assumptions or premises which occur outside the proof box opened by the dummy variable x_0. These restrictions (a side condition) imply that the case we have for ϕ is, after all, quite general.

To understand this, think of the following analogy. If you want to prove to someone that you can (say) split a tennis ball in your hand by squashing it, you might say 'OK, give me a tennis ball and I'll split it'. So we give you one and you do it. But how can we be sure that you could split *any* tennis ball in this way? Of course, we can't give you *all of them*, so how could we be sure that you could split any one? Well, we assume that the one you did split was an arbitrary, or 'random', one, i.e. that it wasn't special in any way (like a ball which you had 'prepared' beforehand); and that is enough to convince us that you could split *any* tennis ball. Our rule says that if you can prove ϕ about an x_0 that isn't special in any way, then you could prove it for any x whatsoever.

To put it another way, the step from ϕ to $\forall x\,\phi$ is legitimate only if we

have arrived at ϕ in such a way that none of its assumptions contain x as a free variable. Any assumption which has a free occurrence of x puts constraints on such an x. For example, the assumption bird(x) confines x to the realm of birds and anything we can prove about x using this formula will have to be a statement restricted to birds and not about anything else we might have had in mind.

It is time we looked at an example of these rules at work. Here is a proof of the sequent

$$\forall x\,(P(x) \to Q(x)), \forall x\,P(x) \vdash \forall x\,Q(x):$$

1		$\forall x\,(P(x) \to Q(x))$	premise
2		$\forall x\,P(x)$	premise
3	x_0	$P(x_0) \to Q(x_0)$	$\forall x$ e 1
4		$P(x_0)$	$\forall x$ e 2
5		$Q(x_0)$	$\to$e 3, 4
6		$\forall x\,Q(x)$	$\forall x$ i 3–5

The structure of this proof is guided by the fact that the conclusion is a $\forall$ formula. To arrive at this, we will need an application of $\forall x$ i, so we set up the box controlling the scope of x_0. The rest is now mechanical: we prove $\forall x\,Q(x)$ by proving $Q(x_0)$; but the latter we can prove as soon as we can prove $P(x_0)$ and $P(x_0) \to Q(x_0)$, which themselves are instances of the premises (obtained by $\forall$e with the term x_0). Note that we wrote the name of the dummy variable to the left of the first proof line in its scope box.

Here is a simpler example which uses only $\forall x$ e: we show the sequent

$$P(t), \forall x\,(P(x) \to \neg Q(x)) \vdash \neg Q(t)$$

for any term t:

1	$P(t)$	premise
2	$\forall x\,(P(x) \to \neg Q(x))$	premise
3	$P(t) \to \neg Q(t)$	$\forall x$ e 2
4	$\neg Q(t)$	$\to$e 3, 1

Note that we invoked $\forall x$ e with the same instance t as in the assumption $P(t)$. If we had invoked $\forall x$ e with y, say, and obtained $P(y) \to \neg Q(y)$, then that would have been valid, but it would not have been helpful in the case

that y was different from t. Thus, $\forall x\,\mathrm{e}$ is really a *scheme* of rules, one for each term t (free for x in ϕ), and we should make our choice on the basis of consistent pattern matching. Further, note that we have rules $\forall x\,\mathrm{i}$ and $\forall x\,\mathrm{e}$ *for each variable* x. In particular, there are rules $\forall y\,\mathrm{i}$, $\forall y\,\mathrm{e}$ and so on. We will write $\forall\mathrm{i}$ and $\forall\mathrm{e}$ when we speak about such rules without concern for the actual quantifier variable.

Notice also that, although the square brackets representing substitution appear in the rules $\forall\mathrm{i}$ and $\forall\mathrm{e}$, they do not appear when we use those rules. The reason for this is that we actually carry out the substitution that is asked for. In the rules, the expression $\phi[t/x]$ means: 'ϕ, but with free occurrences of x replaced by t'. Thus, if ϕ is $P(x,y) \to Q(y,z)$ and the rule refers to $\phi[a/y]$, we carry out the substitution and write $P(x,a) \to Q(a,z)$ in the proof.

A helpful way of understanding the universal quantifier rules is to compare the rules for $\forall$ with those for $\wedge$. The rules for $\forall$ are in some sense generalisations of those for $\wedge$; whereas $\wedge$ has just two conjuncts, $\forall$ acts like it conjoins lots of formulas (one for each substitution instance of its variable). Thus, whereas $\wedge\mathrm{i}$ has two premises, $\forall x\,\mathrm{i}$ has a premise $\phi[x_0/x]$ for each possible 'value' of x_0. Similarly, where and-elimination allows you to deduce from $\phi \wedge \psi$ whichever of ϕ and ψ you like, forall-elimination allows you to deduce $\phi[t/x]$ from $\forall x\,\phi$, for whichever t you like. To say the same thing another way: think of $\forall x\,\mathrm{i}$ as saying: to prove $\forall x\,\phi$, you have to prove $\phi[x_0/x]$ for every possible value x_0; while $\wedge\mathrm{i}$ says that to prove $\phi_1 \wedge \phi_2$ you have to prove ϕ_i for every i.

The proof rules for existential quantification

The analogy between $\forall$ and $\wedge$ extends also to $\exists$ and $\vee$; and you could even try to guess the rules for $\exists$ by starting from the rules for $\vee$ and applying the same ideas as those that related $\wedge$ to $\forall$. For example, we saw that the rules for or-introduction were a sort of dual of those for and-elimination; to emphasise this point, we could write them as

$$\frac{\phi_1 \wedge \phi_2}{\phi_k}\ \wedge\mathrm{e}_k \qquad \frac{\phi_k}{\phi_1 \vee \phi_2}\ \vee\mathrm{i}_k,$$

where k can be chosen to be either 1 or 2.

Therefore, given the form of forall-elimination, we can infer that exists-introduction must be simply

$$\frac{\phi[t/x]}{\exists x\phi}\ \exists x\,\mathrm{i}.$$

Indeed, this is correct: it simply says that we can deduce $\exists x\,\phi$ whenever we

have $\phi[t/x]$ for some term t (naturally, we impose the side condition that t be free for x in ϕ).

In the rule ∃i, we see that the formula $\phi[t/x]$ contains, from a computational point of view, more information than $\exists x\,\phi$. The latter merely says that ϕ holds for some, unspecified, value of x; whereas $\phi[t/x]$ has a witness t at its disposal. Recall that the square-bracket notation asks us actually to carry out the substitution. However, the notation $\phi[t/x]$ is somewhat misleading since it suggests not only the right witness t but also the formula ϕ itself. For example, consider the situation in which t equals y such that $\phi[y/x]$ is $y = y$. Then you can check for yourself that ϕ could be a number of things, like $x = x$ or $x = y$. Thus, $\exists x\,\phi$ will depend on which of these ϕ you were thinking of.

Extending the analogy between ∃ and ∨, the rule ∨e leads us to the following formulation of ∃e:

$$\dfrac{\exists x\,\phi \qquad \boxed{\begin{array}{ll} x_0 & \phi[x_0/x] \\ & \vdots \\ & \chi \end{array}}}{\chi}\ \exists e$$

Like ∨e, it is a case analysis. The reasoning goes: we know $\exists x\,\phi$, so ϕ is true for at least one 'value' of x. So we do a case analysis over all those possible values, writing x_0 as a generic value representing them all. If assuming $\phi[x_0/x]$ allows us to prove some χ which doesn't mention x_0, then this χ must be true whichever x_0 it was. And that's precisely what the rule ∃e allows us to deduce. Of course, we impose the side condition that x_0 can't occur outside the box (therefore, in particular, it cannot occur in χ). The box is controlling two things: the scope of x_0 and also the scope of the assumption $\phi[x_0/x]$.

Just as ∨e says that to use $\phi_1 \vee \phi_2$, you have to be prepared for either of the ϕ_i, so ∃e says that to use $\exists x\,\phi$ you have to be prepared for any possible $\phi[x_0/x]$. Another way of thinking about ∃e goes like this: if you know $\exists x\,\phi$ and you can derive some χ from $\phi[x_0/x]$, i.e. by giving a name to the thing you know exists, then you can derive χ even without giving it the name.

The rule $\exists x\,$e is also similar to ∨e in the sense that both of them are elimination rules which don't have to conclude a *subformula* of the formula they are about to eliminate. Please verify that all other elimination rules so far have this *subformula property*[1]. This property is computationally very pleasant, for it allows us to narrow down the search space for a

[1] For $\forall x\,$e we perform a substitution $[t/x]$, but it preserves the logical structure of ϕ.

proof dramatically. Unfortunately, $\exists x\,\mathrm{e}$, like its cousin $\vee\mathrm{e}$, is not of that computationally benign kind.

Let us practice these rules on a couple of examples. Certainly, we should be able to prove

$$\forall x\,\phi \vdash \exists x\,\phi.$$

In the proof

1	$\forall x\,\phi$	premise
2	$\phi[x/x]$	$\forall x\,\mathrm{e}\ 1$
3	$\exists x\,\phi$	$\exists x\,\mathrm{i}\ 2$

we chose t to be x with respect to both $\forall x\,\mathrm{e}$ and to $\exists x\,\mathrm{i}$ (and note that x is free for x in ϕ and that $\phi[x/x]$ is simply ϕ again).

A more complicated example is the sequent

$$\forall x\,(P(x) \to Q(x)), \exists x\,P(x) \vdash \exists x\,Q(x)$$

which can be proved by

1		$\forall x\,(P(x) \to Q(x))$	premise
2		$\exists x\,P(x)$	premise
3	x_0	$P(x_0)$	assumption
4		$P(x_0) \to Q(x_0)$	$\forall x\,\mathrm{e}\ 1$
5		$Q(x_0)$	$\to\mathrm{e}\ 4, 3$
6		$\exists x\,Q(x)$	$\exists x\,\mathrm{i}\ 5$
7		$\exists x\,Q(x)$	$\exists x\,\mathrm{e}\ 2, 3{-}6$

The motivation for introducing the box in line 3 of this proof is the existential quantifier in the premise $\exists x\,P(x)$ which has to be eliminated. Notice that the $\exists$ in the conclusion has to be introduced *within the box* and observe the nesting of these two steps. The formula $\exists x\,Q(x)$ in line 6 is the instantiation of χ in the rule $\exists\mathrm{e}$ and it is easy to check that it does not contain an occurrence of x_0, as required by the condition that there be no x_0 outside the box.

The almost identical 'proof'

1		$\forall x\,(P(x) \to Q(x))$	premise
2		$\exists x\,P(x)$	premise
3	x_0	$P(x_0)$	assumption
4		$P(x_0) \to Q(x_0)$	$\forall x$ e 1
5		$Q(x_0)$	$\to$e 4, 3
6		$Q(x_0)$	$\exists x$ e 2, 3–5
7		$\exists x\,Q(x)$	$\exists x$ i 6

is not a legal proof; line 6 allows the fresh parameter x_0 to escape the scope of the box which declares it. This is not permissible and we will see on page 120 an example where such illicit use of proof rules results in unsound arguments.

A sequent with a slightly more complex proof is

$$\forall x\,(Q(x) \to R(x)),\ \exists x\,(P(x) \wedge Q(x)) \vdash \exists x\,(P(x) \wedge R(x))$$

which could model some argument such as

If all quakers are reformists and if there is a protestant who is also a quaker, then there must be a protestant who is also a reformist.

One possible proof strategy is to assume $P(x_0) \wedge Q(x_0)$, get the instance $Q(x_0) \to R(x_0)$ from $\forall x\,(Q(x) \to R(x))$ and use $\wedge e_2$ to get our hands on $Q(x_0)$, which gives us $R(x_0)$ via $\to$e ... :

1		$\forall x\,(Q(x) \to R(x))$	premise
2		$\exists x\,(P(x) \wedge Q(x))$	premise
3	x_0	$P(x_0) \wedge Q(x_0)$	assumption
4		$Q(x_0) \to R(x_0)$	$\forall x$ e 1
5		$Q(x_0)$	$\wedge e_2$ 3
6		$R(x_0)$	$\to$e 4, 5
7		$P(x_0)$	$\wedge e_1$ 3
8		$P(x_0) \wedge R(x_0)$	$\wedge$i 7, 6
9		$\exists x\,(P(x) \wedge R(x))$	$\exists x$ i 8
10		$\exists x\,(P(x) \wedge R(x))$	$\exists x$ e 2, 3–9

Note the strategy of this proof: We list the two premises. The second premise is of use here only if we apply $\exists x\,\mathrm{e}$ to it. This sets up the proof box in lines 3–9 as well as the fresh parameter name x_0. Since we want to prove $\exists x\,(P(x) \wedge R(x))$, this formula has to be the last one in the box (our goal) and the rest involves $\forall x\,\mathrm{e}$ and $\exists x\,\mathrm{i}$.

The rules $\forall$i and $\exists$e both have the side condition that the dummy variable cannot occur outside the box in the rule. Of course, these rules may still be nested, by choosing another name (e.g. y_0) for the dummy variable. For example, we will prove the sequent

$$\exists x\,P(x),\ \forall x\,\forall y\,(P(x) \to Q(y)) \vdash \forall y\,Q(y).$$

(Look how strong the second premise is, by the way: given any x, y, if $P(x)$, then $Q(y)$. This means that, if there is any object with the property P, then all objects shall have the property Q.) The proof goes as follows: we take an arbitrary y_0 and prove $Q(y_0)$; this we do by observing that, since some x satisfies P, so by the second premise any y satisfies Q:

1		$\exists x\,P(x)$	premise
2		$\forall x \forall y\,(P(x) \to Q(y))$	premise
3	y_0		
4	x_0	$P(x_0)$	assumption
5		$\forall y\,(P(x_0) \to Q(y))$	$\forall x\,\mathrm{e}\ 2$
6		$P(x_0) \to Q(y_0)$	$\forall y\,\mathrm{e}\ 5$
7		$Q(y_0)$	$\to\mathrm{e}\ 6,4$
8		$Q(y_0)$	$\exists x\,\mathrm{e}\ 1,4\text{–}7$
9		$\forall y\,Q(y)$	$\forall y\,\mathrm{i}\ 3\text{–}8$

There is no special reason for picking x_0 as a name for the dummy variable we use for $\forall x$ and $\exists x$ and y_0 as a name for $\forall y$ and $\exists y$. We do this only because it makes it easier for us humans. Again, study the strategy of this proof. We ultimately have to show a $\forall y$ formula which requires us to use $\forall y\,\mathrm{i}$, i.e. we need to open up a proof box (lines 3–8) whose subgoal is to prove a generic instance $Q(y_0)$. Within that box we want to make use of the premise $\exists x\,P(x)$ which results in the proof box set-up of lines 4–7. Notice that, in line 8, we may well move $Q(y_0)$ out of the box controlled by x_0.

We have emphasised the point that the dummy variables in the rules $\exists$e and $\forall$i must not occur outside their boxes. Here is an example which shows how things would go wrong if we didn't have this side condition. We could

prove the sequent

$$\exists x\, P(x),\ \forall x\,(P(x) \to Q(x)) \vdash \forall y\, Q(y)$$

which is intuitively unsound. (Compare it with the previous sequent; the second premise is now much weaker, allowing us to conclude Q only for those objects for which we know P.) Here is an alleged 'proof':

1		$\exists x\, P(x)$	premise
2		$\forall x\,(P(x) \to Q(x))$	premise
3	x_0		
4	x_0	$P(x_0)$	assumption
5		$P(x_0) \to Q(x_0)$	$\forall x$ e 2
6		$Q(x_0)$	$\to$e 5, 4
7		$Q(x_0)$	$\exists x$ e 1, 4–6
8		$\forall y\, Q(y)$	$\forall y$ i 3–7

The last step introducing $\forall y$ is *not* the bad one; that step is fine. The bad one is the second from last one, concluding $Q(x_0)$ by $\exists x$ e and violating the side condition that x_0 may not leave the scope of its box. You can try a few other ways of 'proving' this sequent, but none of them should work (assuming that our proof system is sound with respect to semantic entailment, which we define in the next section). Without this side condition, we would also be able to prove that 'all x satisfy the property P as soon as one of them does so', a semantic disaster of biblical proportions!

2.3.2 Quantifier equivalences

We have already hinted at semantic equivalences between certain forms of quantification. Now we want to provide formal proofs for some of the most commonly used quantifier equivalences. Quite a few of them involve several quantifications over more than just one variable. Thus, this topic is also good practice for using the proof rules for quantifiers in a nested fashion.

For example, the formula $\forall x \forall y\, \phi$ should be equivalent to $\forall y \forall x\, \phi$ since both say that ϕ should hold for all values of x and y. What about $(\forall x\, \phi) \wedge (\forall x\, \psi)$ versus $\forall x\,(\phi \wedge \psi)$? A moment's thought reveals that they should have the same meaning as well. But what if the second conjunct does not start with $\forall x$? So what if we are looking at $(\forall x\, \phi) \wedge \psi$ in general and want to compare it with $\forall x\,(\phi \wedge \psi)$? Here we need to be careful, since x might be free in ψ and would then become bound in the formula $\forall x\,(\phi \wedge \psi)$.

Here are some quantifier equivalences which you should become familiar with. (Recall that we wrote $\phi_1 \dashv\vdash \phi_2$ in Chapter 1 as an abbreviation for $\phi_1 \vdash \phi_2$ and $\phi_2 \vdash \phi_1$.)

Theorem 2.9 *Let ϕ and ψ be formulas of predicate logic. Then we have the following equivalences:*

1. (a) $\neg\forall x\,\phi \dashv\vdash \exists x\,\neg\phi$
   (b) $\neg\exists x\,\phi \dashv\vdash \forall x\,\neg\phi$.
2. *Assuming that x is* not *free in ψ:*
   (a) $\forall x\,\phi \wedge \psi \dashv\vdash \forall x\,(\phi \wedge \psi)$
   Remember that $\forall x\,\phi \wedge \psi$ is implicitly bracketed as $(\forall x\,\phi) \wedge \psi$, by virtue of the binding priorities.
   (b) $\forall x\,\phi \vee \psi \dashv\vdash \forall x\,(\phi \vee \psi)$
   (c) $\exists x\,\phi \wedge \psi \dashv\vdash \exists x\,(\phi \wedge \psi)$
   (d) $\exists x\,\phi \vee \psi \dashv\vdash \exists x\,(\phi \vee \psi)$
   (e) $\forall x\,(\psi \to \phi) \dashv\vdash \psi \to \forall x\,\phi$
   (f) $\exists x\,(\phi \to \psi) \dashv\vdash \forall x\,\phi \to \psi$.
   (g) $\exists x\,(\psi \to \phi) \dashv\vdash \psi \to \exists x\,\phi$
   (h) $\forall x\,(\phi \to \psi) \dashv\vdash \exists x\,\phi \to \psi$.
3. (a) $\forall x\,\phi \wedge \forall x\,\psi \dashv\vdash \forall x\,(\phi \wedge \psi)$
   Remember that $\forall x\,\phi \wedge \forall x\,\psi$ is implicitly bracketed as $(\forall x\,\phi) \wedge (\forall x\,\psi)$, by virtue of the binding priorities.
   (b) $\exists x\,\phi \vee \exists x\,\psi \dashv\vdash \exists x\,(\phi \vee \psi)$.
4. (a) $\forall x\,\forall y\,\phi \dashv\vdash \forall y\,\forall x\,\phi$
   (b) $\exists x\,\exists y\,\phi \dashv\vdash \exists y\,\exists x\,\phi$.

PROOF: We will prove most of these sequents; the proofs for the remaining ones are straightforward adaptations and are left as exercises. Recall that we sometimes write $\bot$ to denote any contradiction.

1. (a) We will lead up to this by proving two simpler sequents first: $\neg(p_1 \wedge p_2) \vdash \neg p_1 \vee \neg p_2$ and then $\neg\forall x\,P(x) \vdash \exists x\,\neg P(x)$. The reason for proving the first of these is to illustrate the close relationship between $\wedge$ and $\vee$ on the one hand and $\forall$ and $\exists$ on the other — think of a model with just two elements 1 and 2 such that p_i $(i = 1, 2)$ stands for $P(x)$ evaluated at i. The idea is that proving this propositional sequent should give us inspiration for proving the second one of predicate logic. The reason for proving the latter sequent is that it is a special case (in which ϕ equals $P(x)$) of the one we are really after, so

again it should be simpler while providing some inspiration. So, let's go.

1	$\neg(p_1 \wedge p_2)$			premise
2	$\neg(\neg p_1 \vee \neg p_2)$			assumption
3	$\neg p_1$	assumption	$\neg p_2$	assumption
4	$\neg p_1 \vee \neg p_2$	$\vee i_1$ 3	$\neg p_1 \vee \neg p_2$	$\vee i_2$ 3
5	$\bot$	$\neg e$ 4, 2	$\bot$	$\neg e$ 4, 2
6	p_1	RAA 3–5	p_2	RAA 3–5
7	$p_1 \wedge p_2$			$\wedge i$ 6, 6
8	$\bot$			$\neg e$ 7, 1
9	$\neg p_1 \vee \neg p_2$			RAA 2–8

You have seen this sort of proof before, in Chapter 1. It is an example of something which requires proof by contradiction, or $\neg\neg e$, or LEM (meaning that it simply cannot be proved in the reduced natural deduction system which discards these three rules) — in fact, we have used the rule RAA three times.

Now we prove $\neg\forall x\, P(x) \vdash \exists x\, \neg P(x)$ similarly, except that where the rules for $\wedge$ and $\vee$ were used we now use those for $\forall$ and $\exists$:

1	$\neg\forall x\, P(x)$	premise
2	$\neg\exists x\, \neg P(x)$	assumption
3	x_0	
4	$\neg P(x_0)$	assumption
5	$\exists x \neg P(x)$	$\exists x$ i 4
6	$\bot$	$\neg e$ 5, 2
7	$P(x_0)$	RAA 4–6
8	$\forall x\, P(x)$	$\forall x$ i 3–7
9	$\bot$	$\neg e$ 8, 1
10	$\exists x\, \neg P(x)$	RAA 2–9

You will really benefit by spending time understanding the way this proof mimics the one above it. This insight is very useful for constructing predicate logic proofs: you first construct a similar propositional proof and then mimic it.

Next we prove $\neg\forall x\,\phi \vdash \exists x\,\neg\phi$:

1		$\neg\forall x\,\phi$	premise
2		$\neg\exists x\,\neg\phi$	assumption
3	x_0		
4		$\neg\phi[x_0/x]$	assumption
5		$\exists x\neg\phi$	$\exists x$ i 4
6		$\bot$	$\neg$e 5, 2
7		$\phi[x_0/x]$	RAA 4–6
8		$\forall x\,\phi$	$\forall x$ i 3–7
9		$\bot$	$\neg$e 8, 1
10		$\exists x\,\neg\phi$	RAA 2–9

The reverse sequent $\exists x\,\neg\phi \vdash \neg\forall x\,\phi$ is more straightforward, for it does not involve proof by contradiction, $\neg\neg$e, or LEM. Unlike its converse, it has a constructive proof which the intuitionists do accept. We could again prove the corresponding propositional sequent, but we leave that as an exercise.

1		$\exists x\,\neg\phi$	assumption
2		$\forall x\,\phi$	assumption
3	x_0		
4		$\neg\phi[x_0/x]$	assumption
5		$\phi[x_0/x]$	$\forall x$ e 2
6		$\bot$	$\neg$e 5, 4
7		$\bot$	$\exists x$ e 1, 3–6
8		$\neg\forall x\,\phi$	$\neg$i 2–7

2. (a) The sequent $\forall x\,\phi \wedge \psi \vdash \forall x(\phi \wedge \psi)$ can be proved thus:

1		$(\forall x\,\phi) \wedge \psi$	premise
2		$\forall x\,\phi$	$\wedge e_1$ 1
3		ψ	$\wedge e_2$ 1
4	x_0		
5		$\phi[x_0/x]$	$\forall x$ e 2
6		$\phi[x_0/x] \wedge \psi$	$\wedge$i 5, 3
7		$(\phi \wedge \psi)[x_0/x]$	identical to 6, since x not free in ψ
8		$\forall x(\phi \wedge \psi)$	$\forall x$ i 4–7

The reverse argument can go like this:

1		$\forall x(\phi \wedge \psi)$	premise
2	x_0		
3		$(\phi \wedge \psi)[x_0/x]$	$\forall x$ e 1
4		$\phi[x_0/x] \wedge \psi$	identical to 3, since x not free in ψ
5		ψ	$\wedge e_2$ 3
6		$\phi[x_0/x]$	$\wedge e_1$ 3
7		$\forall x\,\phi$	$\forall x$ i 2–6
8		$(\forall x\,\phi) \wedge \psi$	$\wedge$i 7, 5

Notice that the use of $\wedge$i in the last line is permissible, because ψ was obtained for any instantiation of the formula in line 1.

3. (b) The sequent $(\exists x\,\phi) \vee (\exists x\,\psi) \vdash \exists x(\phi \vee \psi)$ has to be proved using the rule $\vee$e; so we have two principal cases, each of which requires the rule $\exists x$ i:

1		$(\exists x\,\phi) \vee (\exists x\,\psi)$			premise
2		$\exists x\,\phi$		$\exists x\,\psi$	assumpt.
3	x_0	$\phi[x_0/x]$	x_0	$\psi[x_0/x]$	assumpt.
4		$\phi[x_0/x] \vee \psi[x_0/x]$		$\phi[x_0/x] \vee \psi[x_0/x]$	$\vee$i 3
5		$(\phi \vee \psi)[x_0/x]$		$(\phi \vee \psi)[x_0/x]$	identical
6		$\exists x(\phi \vee \psi)$		$\exists x(\phi \vee \psi)$	$\exists x$ i 5
7		$\exists x(\phi \vee \psi)$		$\exists x(\phi \vee \psi)$	$\exists x$ e 2, 3–6
8		$\exists x(\phi \vee \psi)$			$\vee$e 1, 2–7

The converse assumes $\exists x(\phi \vee \psi)$ so its proof has to use $\exists x\,\mathrm{e}$ as its last rule; for that rule, we need to assume $\phi \vee \psi$ as a temporary assumption and need to conclude $(\exists x\,\phi) \vee (\exists x\,\psi)$ from those data; of course, the assumption $\phi \vee \psi$ requires the usual case analysis:

1		$\exists x(\phi \vee \psi)$		premise
2	x_0	$(\phi \vee \psi)[x_0/x]$		assumption
3		$\phi[x_0/x] \vee \psi[x_0/x]$		identical
4		$\phi[x_0/x]$	$\psi[x_0/x]$	assumption
5		$\exists x\,\phi$	$\exists x\,\psi$	$\exists x\,\mathrm{i}\ 4$
6		$\exists x\,\phi \vee \exists x\,\psi$	$\exists x\,\phi \vee \exists x\,\psi$	$\vee\mathrm{i}\ 5$
7		$\exists x\,\phi \vee \exists x\,\psi$		$\vee\mathrm{e}\ 3, 4\text{–}6$
8		$\exists x\,\phi \vee \exists x\,\psi$		$\exists x\,\mathrm{e}\ 1, 2\text{–}7$

4. (b) In assuming $\exists x\,\exists y\,\phi$, we have to nest $\exists x\,\mathrm{e}$ and $\exists y\,\mathrm{e}$ to conclude $\exists y\,\exists x\,\phi$. Of course, we have to obey the format of these elimination rules as done below:

1		$\exists x\,\exists y\,\phi$	premise
2	x_0	$(\exists y\,\phi)[x_0/x]$	assumption
3		$\exists y\,(\phi[x_0/x])$	identical, since x, y different variables
4	y_0	$\phi[x_0/x][y_0/y]$	assumption
5		$\phi[y_0/y][x_0/x]$	identical, since x, y, x_0, y_0 different variables
6		$\exists x\,\phi[y_0/y]$	$\forall x\,\mathrm{i}\ 5$
7		$\exists y\,\exists x\,\phi$	$\forall y\,\mathrm{i}\ 6$
8		$\exists y\,\exists x\,\phi$	$\exists y\,\mathrm{e}3, 4\text{–}7$
9		$\exists y\,\exists x\,\phi$	$\exists x\,\mathrm{e}1, 2\text{–}8$

The converse is proven in the same way by swapping the roles of x and y.

□

EXERCISES 2.5

1. The rules for $\forall$ are very similar to those for $\wedge$ and those for $\exists$ are just like those for $\vee$.

(a) Find a (propositional) proof for $\phi \to (q_1 \wedge q_2) \vdash (\phi \to q_1) \wedge (\phi \to q_2)$.

(b) Find a (predicate) proof for $\phi \to \forall x\, Q(x) \vdash \forall x\, (\phi \to Q(x))$, provided that x is not free in ϕ.
(Hint: whenever you used $\wedge$ rules in the (propositional) proof of the previous item, use $\forall$ rules in the (predicate) proof.)

(c) Find a proof for $\forall x\, (P(x) \to Q(x)) \vdash \forall x\, P(x) \to \forall x\, Q(x)$.
(Hint: try $(p_1 \to q_1) \wedge (p_2 \to q_2) \vdash p_1 \wedge p_2 \to q_1 \wedge q_2$ first.)

* (d) Prove $\forall x\, (P(x) \wedge Q(x)) \vdash \forall x\, P(x) \wedge \forall x\, Q(x)$.

* (e) Prove $\forall x\, P(x) \vee \forall x\, Q(x) \vdash \forall x\, (P(x) \vee Q(x))$.

* (f) Prove $\exists x\, (P(x) \wedge Q(x)) \vdash \exists x\, P(x) \wedge \exists x\, Q(x)$.

* (g) Prove $\exists x\, F(x) \vee \exists x\, G(x) \vdash \exists x\, (F(x) \vee G(x))$.

(h) Prove $\forall x\, \forall y\, (S(y) \to F(x)) \vdash \exists y S(y) \to \forall x\, F(x)$.

2. What is the propositional logic sequent that corresponds to $\exists x\, \neg\phi \vdash \neg\forall x\, \phi$? Prove it.

3. Provide proofs for the following sequents:

(a) $\forall x\, P(x) \vdash \forall y\, P(y)$; using $\forall x\, P(x)$ as a premise, your proof needs to end with an application of $\forall$i which requires the formula $P(y_0)$.

(b) $\forall x\, (P(x) \to Q(x)) \vdash (\forall x\, \neg Q(x)) \to (\forall x\, \neg P(x))$

(c) $\forall x\, (P(x) \to \neg Q(x)) \vdash \neg(\exists x\, (P(x) \wedge Q(x)))$.

4. The sequents below look a bit tedious, but in proving them you make sure that you really understand how to nest the rules:

* (a) $\forall x\, \forall y\, P(x, y) \vdash \forall u\, \forall v\, P(u, v)$

(b) $\exists x\, \exists y\, F(x, y) \vdash \exists u\, \exists v\, F(u, v)$

* (c) $\exists x\, \forall y\, P(x, y) \vdash \forall y\, \exists x\, P(x, y)$.

5. In the following exercises, involving the proof rules for quantifiers, whenever you use a rule, you should mention how the relevant syntactic restrictions are satisfied.

(a) Prove one direction of 1(b) of Theorem 2.9: $\neg\exists x\, \phi \vdash \forall x\, \neg\phi$.

(b) Prove 2(b), 2(c), 2(d), 2(e) and 2(f) of Theorem 2.9.

(c) Prove 3(a) of Theorem 2.9: $(\forall x\, \phi) \wedge (\forall x\, \psi) \dashv\vdash \forall x\, (\phi \wedge \psi)$; recall that you have to do two separate proofs.

(d) Prove both directions of 4(a) of the last theorem: $\forall x\, \forall y\, \phi \dashv\vdash \forall y\, \forall x\, \phi$.

6. Prove the following sequents in predicate logic, where P and Q are predicates with one argument:

(a) $\forall x\, (\neg P(x) \wedge Q(x)) \vdash \forall x\, (P(x) \to Q(x))$

(b) $\forall x\,(P(x) \wedge Q(x)) \vdash \forall x\,(P(x) \rightarrow Q(x))$
(c) $\exists x\,(\neg P(x) \wedge \neg Q(x)) \vdash \exists x\,(\neg(P(x) \wedge Q(x)))$
(d) $\exists x\,(\neg P(x) \vee Q(x)) \vdash \exists x\,(\neg(P(x) \wedge \neg Q(x)))$.

7. Just like natural deduction proofs for propositional logic, certain things that look easy can be hard to prove for predicate logic. Typically, these involve the $\neg\neg$ rule. The patterns are the same as in propositional logic:

(a) Proving $p \vee q \vdash \neg(\neg p \wedge \neg q)$ is quite easy. Try it.
(b) Show $\exists x\,P(x) \vdash \neg\forall x\,\neg P(x)$.
(c) Proving $\neg(\neg p \wedge \neg q) \vdash p \vee q$ is hard; you have to try to prove $\neg\neg(p \vee q)$ first and then use the $\neg\neg$e rule. Do it.
* (d) Prove $\neg\forall x\,\neg P(x) \vdash \exists x\,P(x)$.
* (e) Prove $\forall x\,\neg P(x) \vdash \neg\exists x\,P(x)$.
* (f) Prove $\neg\exists x\,P(x) \vdash \forall x\,\neg P(x)$.

8. The proofs of the sequents below combine the proof rules for equality and quantifiers. We write $\phi \leftrightarrow \psi$ as an abbreviation for $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$.

* (a) $P(b) \vdash \forall x\,(x = b \rightarrow P(x))$
(b) $P(b),\ \forall x \forall y\,(P(x) \wedge P(y) \rightarrow x = y) \vdash \forall x\,(P(x) \leftrightarrow x = b)$
* (c) $\exists x\,\exists y\,(H(x, y) \vee H(y, x)),\ \neg\exists x\,H(x, x) \vdash \exists x \exists y\,\neg(x = y)$
(d) $\forall x\,(P(x) \leftrightarrow x = b) \vdash P(b) \wedge \forall x \forall y\,(P(x) \wedge P(y) \rightarrow x = y)$.

9. Prove the following sequents in predicate logic:

* (a) $S \rightarrow \forall x\,Q(x) \vdash \forall x\,(S \rightarrow Q(x))$ (S is a predicate with zero arguments.)
* (b) $\exists x\,(S \rightarrow Q(x)) \vdash S \rightarrow \exists x\,Q(x)$
(c) $S \rightarrow \exists x\,Q(x) \vdash \exists x\,(S \rightarrow Q(x))$
(d) $\exists x\,P(x) \rightarrow S \vdash \forall x\,(P(x) \rightarrow S)$
* (e) $\forall x\,P(x) \rightarrow S \vdash \exists x\,(P(x) \rightarrow S)$
(f) $\forall x\,(P(x) \vee Q(x)) \vdash \forall x\,P(x) \vee \exists x\,Q(x)$
(g) $\forall x\,\exists y\,(P(x) \vee Q(y)) \vdash \exists y\,\forall x\,(P(x) \vee Q(y))$.

10. Show by natural deduction:

* (a) $\forall x\,P(a, x, x),\ \forall x\,\forall y\,\forall z\,(P(x, y, z) \rightarrow P(f(x), y, f(z)))$
$\vdash P(f(a), a, f(a))$
* (b) $\forall x\,P(a, x, x),\ \forall x\,\forall y\,\forall z\,(P(x, y, z) \rightarrow P(f(x), y, f(z)))$
$\vdash \exists z\,P(f(a), z, f(f(a)))$

* (c) $\forall y\, Q(b, y), \forall x \forall y\, (Q(x, y) \rightarrow Q(s(x), s(y))) \vdash \exists z\, (Q(b, z) \wedge Q(z, s(s(b))))$.

11. Prove the following sequents in predicate logic:
    (a) $\forall x \forall y \forall z\, (S(x, y) \wedge S(y, z) \rightarrow S(x, z)), \forall x \neg S(x, x) \vdash \forall x \forall y\, (S(x, y) \rightarrow \neg S(y, x))$
    (b) $\forall x\, (P(x) \vee Q(x)), \exists x \neg Q(x), \forall x\, (R(x) \rightarrow \neg P(x)) \vdash \exists x \neg R(x)$
    (c) $\forall x\, (P(x) \rightarrow (Q(x) \vee R(x))), \neg \exists x\, (P(x) \wedge R(x)) \vdash \forall x\, (P(x) \rightarrow Q(x))$
    (d) $\exists x \exists y\, (S(x, y) \vee S(y, x)) \vdash \exists x \exists y\, S(x, y)$
    (e) $\exists x\, (P(x) \wedge Q(x)), \forall y\, (P(x) \rightarrow R(x)) \vdash \exists x\, (R(x) \wedge Q(x))$.
12. Translate the following argument into a sequent in predicate logic using a suitable set of predicate symbols:

    If there are any tax payers, then all politicians are tax payers. If there are any philanthropists, then all tax payers are philanthropists. So, if there are any tax-paying philanthropists, then all politicians are philanthropists.

    Now come up with a proof of that sequent in predicate logic.
13. Discuss in what sense the equivalences of Theorem 2.9 form the basis of an algorithm which pushes quantifiers to the top of a formula's parse tree.

2.4 Semantics of predicate logic

Having seen how natural deduction of propositional logic can be extended to the predicate case, let's now look at how the semantics of predicate logic works. Just like in the propositional case, the semantics should provide a separate, but ultimately equivalent, characterisation of the logic. By 'separate', we mean that the meaning of the connectives is defined in a different way; in proof theory, they were defined by proof rules providing an *operative* explanation. In semantics, we expect something like truth tables. By 'equivalent', we mean that we should be able to prove soundness and completeness, as we did for propositional logic (although a fully fledged proof of soundness and completeness for predicate logic is beyond the scope of this book).

Before we begin describing the semantics of predicate logic, let us look more closely at the real difference between a semantic and a proof-theoretic account. In proof theory, the basic object which is constructed is a proof. Let us write Γ as a shorthand for lists of formulas $\phi_1, \phi_2, \ldots, \phi_n$. Thus, to show that $\Gamma \vdash \phi$, we need to provide a proof of ϕ from Γ. Yet, how can we show that ϕ is not a consequence of Γ? Intuitively, this is harder; how can

you possibly show that *there is no proof* of something? You would have to consider every 'candidate' proof and show it is not one. Thus, proof theory gives a 'positive' characterisation of the logic; it provides convincing evidence for assertions like $\Gamma \vdash \phi$, but it is not very useful for establishing $\Gamma \nvdash \phi$.

Semantics, on the other hand, works in the opposite way. To show that ϕ is *not* a consequence of Γ is the easy bit: you simply give a model of Γ which is not a model of ϕ. Showing that ϕ is a consequence of Γ, on the other hand, is harder in principle. For propositional logic, you need to show that every valuation (an assignment of truth values to all atoms involved) that makes Γ true also makes ϕ true. If there is a small number of valuations, this is not so bad. However, when we look at predicate logic, we will find that there are infinitely many models to consider (the notion corresponding to the valuation in propositional logic is called a model). Thus, in semantics, we have a 'negative' characterisation of the logic. We find establishing assertions of the form $\Gamma \nvDash \phi$ (ϕ is not a semantic entailment of all formulas in Γ) easier than establishing $\Gamma \vDash \phi$ (ϕ is a semantic entailment of Γ), for in the former case we need only talk about one model, whereas in the latter we have to talk about infinitely many.

All this goes to show that it is important to study *both* proof theory *and* semantics. For example, if you are trying to show that ϕ is not a consequence of Γ and you have a hard time doing that, you might want to change your strategy for a while by trying to prove $\Gamma \vdash \phi$. If you find a proof, you know for sure that ϕ is a consequence of Γ. If you can't find a proof, then your attempts at proving it often provide insights which lead you to the construction of a counter example. The fact that proof theory and semantics are equivalent is amazing, but it does not stop them having separate roles in logic, each meriting close study.

2.4.1 Models

Recall how we evaluated formulas in propositional logic. For example, given the propositional formula

$$(p \vee \neg q) \rightarrow (q \rightarrow p)$$

we evaluated this expression by computing a truth value (`T` or `F`) for it, based on a given valuation (assumed truth values for p and q). This activity is essentially the construction of one line in the truth table of $(p \vee \neg q) \rightarrow (q \rightarrow p)$. How can we evaluate formulas in predicate logic? We 'enrich' the formula above to

$$\forall x \, \exists y \, ((P(x) \vee \neg Q(y)) \rightarrow (Q(x) \rightarrow P(y))).$$

Could we simply assume truth values for $P(x)$, $Q(y)$, $Q(x)$ and $P(y)$ and compute a truth value as before? Not quite, since we have to reflect the meaning of the quantifiers $\forall x$ and $\exists y$, their *dependences* and the actual parameters of P and Q — a formula $\forall x \exists y R(x, y)$ generally means something else other than $\exists y \forall x R(x, y)$; why? The problem is that variables are place holders for any, or some, unspecified concrete value. Such values can be of almost any kind: students, birds, numbers, complicated mathematical objects, data structures, programs and so on.

Thus, if we encounter a formula $\exists y \psi$, we try to find some instance of y (some concrete value) such that ψ holds for that particular instance of y. If this succeeds (i.e. there is such a value of y for which ψ holds), then $\exists y \psi$ evaluates to T; otherwise (i.e. there is *no* concrete value of y which realises ψ) it returns F. Dually, evaluating $\forall x \psi$ amounts to showing that ψ evaluates to T for *all* possible values of x; if this is successful, we know that $\forall x \psi$ evaluates to T; otherwise (i.e. there is *some* value of x such that ψ computes F) it returns F. Of course, such evaluations of formulas require a fixed universe of concrete values, the things we are, so to speak, talking about. Thus, the truth value of a formula in predicate logic depends on, and varies with, the actual choice of values and the meaning of the predicate and function symbols involved.

If variables can take on only finitely many values, we can write a program that evaluates formulas in a compositional way. If the root node of ϕ is $\wedge$, $\vee$, $\rightarrow$ or $\neg$, we can compute the truth value of ϕ by using the truth table of the respective logical connective and by computing the truth values of the subtree(s) of that root, as discussed in Chapter 1. If the root is a quantifier, we have sketched above how to proceed. This leaves us with the case of the root node being a predicate symbol P (in propositional logic this was an atom and we were done already). Such a predicate requires n arguments which have to be terms $t_1, t_2, \ldots, t_n$. Therefore, we need to be able to assign truth values to formulas of the form $P(t_1, t_2, \ldots, t_n)$.

For formulas $P(t_1, t_2, \ldots, t_n)$, there is more going on than in the case of propositional logic. For $n = 2$, the predicate P could stand for something like 'the number computed by t_1 is less than, or equal to, the number computed by t_2'. Therefore, we cannot just assign truth values to P in a random fashion. We require a *model* of all function and predicate symbols involved. For example, terms could denote *real numbers* and P could denote the relation 'less than or equal to' on the set of real numbers.

Definition 2.10 Let $\mathscr{F}$ be a set of function symbols and $\mathscr{P}$ a set of predicate

symbols, each symbol with a fixed number of required arguments. A *model* $\mathcal{M}$ of the pair $(\mathcal{F}, \mathcal{P})$ consists of the following set of data:

1. A non-empty set A, the *universe of concrete values*;
2. for each $f \in \mathcal{F}$ with n arguments a concrete function
$$f^{\mathcal{M}} : A^n \to A$$
from A^n, the set of n-tuples over A, to A; and
3. for each $P \in \mathcal{P}$ with n arguments a subset $P^{\mathcal{M}} \subseteq A^n$ of n-tuples over A.

The distinction between f and $f^{\mathcal{M}}$ and between P and $P^{\mathcal{M}}$ is most important. The symbols f and P are just that: symbols, whereas $f^{\mathcal{M}}$ and $P^{\mathcal{M}}$ denote a concrete function and relation in a model $\mathcal{M}$, respectively.

Example 2.11 Let $\mathcal{F} \stackrel{\text{def}}{=} \{+, *, -\}$ and $\mathcal{P} \stackrel{\text{def}}{=} \{=, \leq, <, zero\}$, where $+$, $*$ and $-$ take two arguments and s one; and where $=$, $\leq$ and $<$ are predicates with two arguments and *zero* is a predicate with just one argument. We choose as a model $\mathcal{M}$ the following:

1. The non-empty set A is the set of real numbers.
2. The functions $+^{\mathcal{M}}$, $*^{\mathcal{M}}$ and $-^{\mathcal{M}}$ take two real numbers as arguments and return their *sum*, *product* and *difference*, respectively.
3. The predicates $=^{\mathcal{M}}$, $\leq^{\mathcal{M}}$ and $<^{\mathcal{M}}$ model the relations *equal to*, *less than* and *strictly less than*, respectively. The predicate $zero^{\mathcal{M}}$ holds for r iff r equals 0.

Example 2.12 Let $\mathcal{F} \stackrel{\text{def}}{=} \{e, \cdot\}$ and $\mathcal{P} \stackrel{\text{def}}{=} \{\leq\}$, where e is a constant, $\cdot$ is a function of two arguments and $\leq$ is a predicate in need of two arguments as well. Again, we write $\cdot$ and $\leq$ in infix notation as in

$$(t_1 \cdot t_2) \leq (t \cdot t).$$

The model $\mathcal{M}$ we have in mind has as set A all binary strings, finite words over the alphabet $\{0, 1\}$, including the empty string denoted by ϵ. The interpretation $e^{\mathcal{M}}$ of e is just the empty word ϵ. The interpretation $\cdot^{\mathcal{M}}$ of $\cdot$ is the concatenation of words. For example, $0110 \cdot^{\mathcal{M}} 1110$ equals 01101110. In general, if $a_1 a_2 \ldots a_k$ and $b_1 b_2 \ldots b_n$ are such words with $a_i, b_j \in \{0, 1\}$, then $a_1 a_2 \ldots a_k \cdot^{\mathcal{M}} b_1 b_2 \ldots b_n$ equals $a_1 a_2 \ldots a_k b_1 b_2 \ldots b_n$. Finally, we interpret $\leq$ as the *prefix ordering* of words. We say that s_1 is a *prefix* of s_2 if there is a binary word s_3 such that $s_1 \cdot^{\mathcal{M}} s_3$ equals s_2. For example, 011 is a prefix of 011001 and 011, but 010 is not. Thus, $\leq^{\mathcal{M}}$ is the set $\{(s_1, s_2) \mid s_1 \text{ is a prefix of } s_2\}$.

Here are some formulas in predicate logic which we want to check on this model informally:

- In our model, the formula

$$\forall x\,((x \leq x \cdot e) \wedge (x \cdot e \leq x))$$

  says that every word is a prefix of itself concatenated with the empty word and conversely. Clearly, this holds in our model, for $s \cdot^{\mathcal{M}} \epsilon$ is just s and every word is a prefix of itself.
- In our model, the formula

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

  says that there exists a word s that is a prefix of every other word. This is true, for we may chose ϵ as such a word (there is no other choice in this case).
- In our model, the formula

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

  says that every word has a prefix. This is clearly the case and there are in general multiple choices for y, which are dependent on x.
- In our model, the formula $\forall x\,\forall y\,\forall z\,((x \leq y) \rightarrow (x \cdot z \leq y \cdot z))$ says that whenever a word s_1 is a prefix of s_2, then $s_1 s$ has to be a prefix of $s_2 s$ for every word s. This is clearly *not* the case. For example, take s_1 as 01, s_2 as 011 and s to be 0.
- In our model, the formula

$$\neg\exists x\,\forall y\,((x \leq y) \rightarrow (y \leq x))$$

  says that there is no word s such that whenever s is a prefix of some other word s_1, it is the case that s_1 is a prefix of s as well. This is true since there cannot be such an s. Assume, for the sake of argument, that there were such a word s. Then s is clearly a prefix of $s0$, but $s0$ cannot be a prefix of s since $s0$ contains one more bit than s.

It is crucial to realise that the notion of a model is extremely liberal and open-ended. All it takes is to choose a non-empty set A, whose elements model real-world objects, and a set of concrete functions and relations, one for each function, respectively predicate, symbol. The only mild requirement imposed on all of this is that the concrete functions and relations on A have the same number of arguments as their syntactic counterparts.

However, you, as a designer or implementor of such a model, have the responsibility of choosing your model wisely. Your model should be a

sufficiently accurate picture of whatever it is you want to model, but at the same time it should abstract away (= ignore) aspects of the world which are irrelevant from the perspective of your task at hand.

For example, if you build a database of family relationships, then it would be foolish to interpret *father-of*(x, y) by something like 'x is the daughter of y'. By the same token, you probably would not want to have a predicate for 'is taller than', since your focus in this model is merely on relationships defined by birth. Of course, there are circumstances in which you may want to add additional features to your database.

Given a model $\mathcal{M}$ for a pair $(\mathcal{F}, \mathcal{P})$ of function and predicate symbols, we are now almost in a position to formally compute a truth value for all formulas in predicate logic which involve only function and predicate symbols from $(\mathcal{F}, \mathcal{P})$. There is still one thing, though, that we need to discuss. Given a formula $\forall x\, \phi$ or $\exists x\, \phi$, we intend to check whether ϕ holds for all, respectively some, value a in our model. While this is intuitive, we have no way of expressing this in our syntax: the formula ϕ usually has x as a free variable; $\phi[a/x]$ is well-intended, but ill-formed since $\phi[a/x]$ is *not* a logical formula, for a is not a term but an element of our model.

Therefore we are forced to interpret formulas *relative to an environment*. You may think of environments in a variety of ways. Essentially, they are look-up tables for all variables; such a table l associates with every variable x a value $l(x)$ of the model. So you can also say that environments are functions

$$l : \mathsf{var} \to A$$

from the set of variables var to the universe of values A of the underlying model. Given such a look-up table, we can assign truth values to all formulas. However, for some of these computations we need *updated* look-up tables.

Definition 2.13 Let l be a look-up table for a universe of concrete values A and let $a \in A$. We denote by $l[x \mapsto a]$ the look-up table which maps x to a and any other variable y to $l(y)$.

Finally, we are able to give a semantics to formulas of predicate logic. For propositional logic, we did this by computing a truth value. Clearly, it suffices to know in which cases this value is T.

Definition 2.14 Given a model $\mathcal{M}$ for a pair $(\mathcal{F}, \mathcal{P})$ and given an environment l, we define the *satisfaction relation*

$$\mathcal{M} \models_l \phi$$

for each logical formula ϕ over the pair $(\mathscr{F}, \mathscr{P})$ by structural induction on ϕ. The denotation $\mathscr{M} \models_l \phi$ says that ϕ computes to T in the model $\mathscr{M}$ with respect to the environment l.

P: If ϕ is of the form $P(t_1, t_2, \ldots, t_n)$, then we interpret the terms $t_1, t_2, \ldots, t_n$ in our set A by replacing all variables with their values according to l. In this way we compute concrete values $a_1, a_2, \ldots, a_n$ of A for each of these terms, where we interpret any function symbol $f \in \mathscr{F}$ by $f^{\mathscr{M}}$. Now $\mathscr{M} \models_l P(t_1, t_2, \ldots, t_n)$ holds iff $(a_1, a_2, \ldots, a_n)$ is in the set $P^{\mathscr{M}}$.

$\forall x$: The relation $\mathscr{M} \models_l \forall x \psi$ holds iff $\mathscr{M} \models_{l[x \to a]} \psi$ holds for all $a \in A$.

$\exists x$: Dually, $\mathscr{M} \models_l \exists x \psi$ holds iff $\mathscr{M} \models_{l[x \to a]} \psi$ holds for some $a \in A$.

$\neg$: The relation $\mathscr{M} \models_l \neg \psi$ holds iff it is not the case that $\mathscr{M} \models_l \psi$ holds.

$\vee$: The relation $\mathscr{M} \models_l \psi_1 \vee \psi_2$ holds iff $\mathscr{M} \models_l \psi_1$ or $\mathscr{M} \models_l \psi_2$ holds.

$\wedge$: The relation $\mathscr{M} \models_l \psi_1 \wedge \psi_2$ holds iff $\mathscr{M} \models_l \psi_1$ and $\mathscr{M} \models_l \psi_2$ hold.

$\rightarrow$: The relation $\mathscr{M} \models_l \psi_1 \rightarrow \psi_2$ holds iff $\mathscr{M} \models_l \psi_2$ holds whenever $\mathscr{M} \models_l \psi_1$ holds.

We sometimes write $\mathscr{M} \not\models_l \phi$ to denote that $\mathscr{M} \models_l \phi$ *does not hold.*

There is a straightforward inductive argument on the height of the parse tree of a formula which says that $\mathscr{M} \models_l \phi$ holds iff $\mathscr{M} \models_{l'} \phi$ holds, whenever l and l' are two environments which are identical on the set of free variables of ϕ. In particular, if ϕ has *no* free variables at all, we then call ϕ a *sentence*; we conclude that $\mathscr{M} \models_l \phi$ holds, or does not hold, regardless of the choice of l. Thus, for sentences ϕ we often write

$$\mathscr{M} \models \phi$$

since the choice of an environment l is then irrelevant.

Example 2.15 Let us illustrate the definitions above by means of another simple example. Let $\mathscr{F} \stackrel{\text{def}}{=} \{\mathsf{alma}\}$ and $\mathscr{P} \stackrel{\text{def}}{=} \{\mathsf{loves}\}$ where alma is a constant and loves a predicate with two arguments. The model $\mathscr{M}$ we choose here consists of the set $A \stackrel{\text{def}}{=} \{a, b, c\}$, the constant function $\mathsf{alma}^{\mathscr{M}} \stackrel{\text{def}}{=} a$ and the predicate

$$\mathsf{loves}^{\mathscr{M}} \stackrel{\text{def}}{=} \{(a, a), (b, a), (c, a)\},$$

which has two arguments as required. We want to check whether the model $\mathscr{M}$ satisfies

None of Alma's lovers' lovers love her.

First, we need to express the, morally worrying, sentence in predicate logic. Here is such an encoding (one is often able to find other encodings which differ slightly from the one that is closest to the linguistic and semantic structure of the sentence):

$$\forall x \forall y\, (\mathsf{loves}(x, \mathsf{alma}) \wedge \mathsf{loves}(y, x) \rightarrow \neg \mathsf{loves}(y, \mathsf{alma})).$$

Does the model $\mathcal{M}$ satisfy this formula? Well, it does not; for we may choose a for x and b for y. Since (a, a) is in the set $\mathsf{loves}^{\mathcal{M}}$ and (b, a) is in the set $\mathsf{loves}^{\mathcal{M}}$, we would need that the latter does not hold since it is the interpretation of $\mathsf{loves}(y, \mathsf{alma})$; this cannot be.

And what changes if we modify $\mathcal{M}$ to $\mathcal{M}'$, where we keep A and $\mathsf{alma}^{\mathcal{M}}$, but redefine the interpretation of loves as

$$\mathsf{loves}^{\mathcal{M}'} \stackrel{\text{def}}{=} \{(b, a), (c, b)\}''?$$

Well, now there is exactly one lover of Alma's lovers, namely c; but c is not one of Alma's lovers. Thus, the formula above holds in the model $\mathcal{M}'$.

EXERCISES 2.6

* 1. Consider the formula

$$\phi \stackrel{\text{def}}{=} \forall x \forall y\, Q(g(x, y), g(y, y), z).$$

Obviously, Q is a predicate with three arguments and g a function with two arguments. Find two models $\mathcal{M}$ and $\mathcal{M}'$ with respective environments l and l' such that $\mathcal{M} \models_l \phi$ but $\mathcal{M}' \not\models_{l'} \phi$.

2. Consider the sentence

$$\phi \stackrel{\text{def}}{=} \forall x \exists y \exists z\, (P(x, y) \wedge P(z, y) \wedge (P(x, z) \rightarrow P(z, x))).$$

Which of the following models satisfies ϕ?

   (a) The model $\mathcal{M}$ consists of the set of natural numbers with $P^{\mathcal{M}} \stackrel{\text{def}}{=} \{(m, n) \mid m < n\}$.

   (b) The model $\mathcal{M}'$ consists of the set of natural numbers with $P^{\mathcal{M}'} \stackrel{\text{def}}{=} \{(m, 2 * m) \mid m \text{ natural number}\}$.

   (c) The model $\mathcal{M}''$ consists of the set of natural numbers with $P^{\mathcal{M}''} \stackrel{\text{def}}{=} \{(m, n) \mid m < n + 1\}$.

3. Let P be a predicate with two arguments. Find a model $\mathcal{M}$ which satisfies the sentence $\forall x \neg P(x, x)$. Find also a model $\mathcal{M}'$ such that $\mathcal{M}' \not\models \forall x \neg P(x, x)$.

4. Consider the sentence $\forall x(\exists y P(x,y) \wedge (\exists z P(z,x) \rightarrow \forall y P(x,y)))$. We already noted that its meaning in a given model is independent of the chosen look-up table l. Please simulate the evaluation of this sentence in a model of your choice, focusing on how the initial look-up table l grows and shrinks like a stack when you evaluate its subformulas according to the definition of the satisfaction relation.
5. Let $\mathcal{F} \stackrel{\text{def}}{=} \{d, f, g\}$, where d is a constant symbol, f a function symbol with three arguments and g a function symbol with two arguments. As model $\mathcal{M}$, we choose the set of natural numbers $0, 1, 2, \ldots$. Further, $d^{\mathcal{M}} \stackrel{\text{def}}{=} 2$, $f^{\mathcal{M}}(k,n,m) \stackrel{\text{def}}{=} k * n + m$ and $g^{\mathcal{M}}(k,n) \stackrel{\text{def}}{=} k + n * n$. E.g. $f^{\mathcal{M}}(1,2,3)$ equals 5 and $g^{\mathcal{M}}(2,3)$ equals 11. Assuming a look-up table l with $l(x) \stackrel{\text{def}}{=} 5$ and $l(y) \stackrel{\text{def}}{=} 7$, compute the meaning of the terms below in the model $\mathcal{M}$:
   * (a) $f(d,x,d)$
   (b) $f(g(x,d),y,g(d,d))$
   (c) $g(f(g(d,y),f(x,g(d,d),x),y),f(y,g(d,d),d))$.
6. Let ϕ be the formula

$$\forall x \, \forall y \, \exists z \, (R(x,y) \rightarrow R(y,z)),$$

   where R is a predicate symbol of two arguments.
   * (a) Let $A \stackrel{\text{def}}{=} \{a,b,c,d\}$ and $R^{\mathcal{M}} \stackrel{\text{def}}{=} \{(b,c),(b,b),(b,a)\}$. Do we have $\mathcal{M} \models \phi$? Justify your answer, whatever it is.
   * (b) Let $A' \stackrel{\text{def}}{=} \{a,b,c\}$ and $R^{\mathcal{M}'} \stackrel{\text{def}}{=} \{(b,c),(a,b),(c,b)\}$. Do we have $\mathcal{M}' \models \phi$? Justify your answer, whatever it is.

2.4.2 *Semantic entailment*

Given a model $\mathcal{M}$ for a formula ϕ and an environment l for $\mathcal{M}$, we have learned how to check whether $\mathcal{M}$ satisfies ϕ with respect to l; the affirmative we denoted as $\mathcal{M} \models_l \phi$. This is strikingly different from what happened in propositional logic. There we had a list of *formulas* on the left-hand side of the sign $\models$. We wrote $\phi_1, \phi_2, \ldots, \phi_n \models \psi$ to express the semantic entailment of ψ from $\phi_1, \phi_2, \ldots, \phi_n$: whenever all $\phi_1, \phi_2, \ldots, \phi_n$ evaluate to T, the formula ψ evaluates to T as well. How can we define such a notion for formulas in predicate logic?

Definition 2.16 Let $\phi_1, \phi_2, \ldots, \phi_n, \psi$ be formulas in predicate logic. Then $\phi_1, \phi_2, \ldots, \phi_n \models \psi$ denotes that, whenever $\mathcal{M} \models_l \phi_i$ for $1 \leq i \leq n$, then $\mathcal{M} \models_l \psi$, *for all* models $\mathcal{M}$ and look-up tables l.

The symbol $\models$ is overloaded in predicate logic. We use it to denote satisfiability:

$$\text{there is some model } \mathcal{M} \text{ with } \mathcal{M} \models \phi$$

of sentences and semantic entailment:

$$\phi_1, \phi_2, \ldots, \phi_n \models \psi$$

of formulas. Computationally, each of these notions means trouble. First, establishing $\mathcal{M} \models \phi$ will cause problems, if done on a machine, as soon as the universe of values A of $\mathcal{M}$ is infinite. For example, if ϕ is a sentence of the form $\forall x\, \psi$, then we need to verify $\mathcal{M} \models_{[x \to a]} \psi$ for infinitely many elements a.

Second, and much more seriously, in trying to verify $\phi_1, \phi_2, \ldots, \phi_n \models \psi$, we have to check things out for *all possible models*, i.e. all models which are equipped with the right structure (i.e. they have functions and predicates with the matching number of arguments). This task is impossible to perform mechanically. This should be contrasted to the situation in propositional logic, where the computation of the truth tables of the propositions involved was the basis for computing this relationship successfully.

However, we can sometimes reason that certain semantic entailments are valid. We do this by providing an argument that does not depend on the actual model at hand. Of course, this works only for a very limited number of cases. The most prominent ones are the *quantifier equivalences* which we already encountered in the section on natural deduction.

Let us look at a couple of examples of semantic entailment.

- The justification of the semantic entailment

$$\forall x\,(P(x) \to Q(x)) \models \forall x\, P(x) \to \forall x\, Q(x)$$

  is as follows. Let $\mathcal{M}$ be a model satisfying $\forall x\,(P(x) \to Q(x))$. We need to show that $\mathcal{M}$ satisfies $\forall x\, P(x) \to \forall x\, Q(x)$ as well. On inspecting the definition of $\mathcal{M} \models \psi_1 \to \psi_2$, we see that we are done if not every element of our model satisfies P. Otherwise, every element does satisfy P. But since $\mathcal{M}$ satisfies $\forall x\,(P(x) \to Q(x))$, the latter fact forces every element of our model to satisfy Q as well. By combining these two cases (i.e. either all elements of $\mathcal{M}$ satisfy P, or not) we have shown that $\mathcal{M}$ satisfies $\forall x\, P(x) \to \forall x\, Q(x)$.
- What about the converse of the above? Is

$$\forall x\, P(x) \to \forall x\, Q(x) \models \forall x\,(P(x) \to Q(x))$$

valid as well? Hardly! Suppose that $\mathcal{M}'$ is a model satisfying $\forall x\, P(x) \to \forall x\, Q(x)$. If A' is its underlying set and $P^{\mathcal{M}'}$ and $Q^{\mathcal{M}'}$ are the corresponding interpretations of P and Q, then $\mathcal{M}' \models \forall x\, P(x) \to \forall x\, Q(x)$ simply says that, if $P^{\mathcal{M}'}$ equals A', then $Q^{\mathcal{M}'}$ must equal A' as well. However, if $P^{\mathcal{M}'}$ does not equal A', then this implication is vacuously true (remember that $\mathtt{F} \to \cdot = \mathtt{T}$ no matter what $\cdot$ actually is). In this case we do not get any additional constraints on our model $\mathcal{M}'$. After these observations, it is now easy to construct a counter example. Let $A' \stackrel{\text{def}}{=} \{a, b\}$, $P^{\mathcal{M}'} \stackrel{\text{def}}{=} \{a\}$ and $Q^{\mathcal{M}'} \stackrel{\text{def}}{=} \{b\}$. Then $\mathcal{M}' \models \forall x\, P(x) \to \forall x\, Q(x)$ holds, but $\mathcal{M}' \models \forall x\, (P(x) \to Q(x))$ does not.

2.4.3 The semantics of equality

We have already pointed out the open-ended nature of the semantics of predicate logic. Given a predicate logic over a set of function symbols $\mathcal{F}$ and a set of predicate symbols $\mathcal{P}$, we need only a non-empty set A equipped with concrete functions $f^{\mathcal{M}}$ (for $f \in \mathcal{F}$) and concrete predicates $P^{\mathcal{M}}$ (for $P \in \mathcal{P}$) in A which have the number of arguments agreed upon in our specification. Of course, we also stressed that most models have natural interpretations of functions and predicates, but notions like *semantic entailment:*

$$\phi_1, \phi_2, \ldots, \phi_n \models \psi$$

really depend on *all possible models*, even the ones that don't seem to make any sense. Apparently there is no way out of this peculiarity. For example, where would you draw the line between a model that makes sense and one that doesn't? And would any such choice, or such a set of criteria, not be *subjective*? Such constraints could also forbid a modification of your model if this alteration were caused by a slight adjustment of the problem domain you intended to model. You see that there are a lot of good reasons for maintaining such a liberal stance towards the notion of models in predicate logic.

However, there is one famous exception. Often one presents predicate logic such that there is always a special predicate $=$ available to denote equality (recall Section 2.3.1); it has two arguments and

$$t_1 = t_2$$

has the intended meaning that the terms t_1 and t_2 compute the same thing. We discussed its proof rule in natural deduction already in Section 2.3.1. Semantically, one recognises the special role of equality by imposing on an interpretation function $=^{\mathcal{M}}$ to be actual equality on the set A. Thus, (a, b) is

in the set $=^{\mathcal{M}}$ iff a and b are the same elements in the set A. For example, given $A \stackrel{\text{def}}{=} \{a, b, c\}$, we are forced to interpret equality such that $=^{\mathcal{M}}$ is just

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

Hence the semantics of equality is easy, for it is always modelled *extensionally*.

EXERCISES 2.7

* 1. Consider the three sentences

$$\begin{aligned} \phi_1 &\stackrel{\text{def}}{=} \forall x\, P(x, x) \\ \phi_2 &\stackrel{\text{def}}{=} \forall x \forall y\, (P(x, y) \rightarrow P(y, x)) \\ \phi_3 &\stackrel{\text{def}}{=} \forall x \forall y \forall z\, ((P(x, y) \wedge P(y, z) \rightarrow P(x, z))) \end{aligned}$$

which express that the binary predicate P is *reflexive, symmetric and transitive*, respectively. Show that none of these sentences is semantically entailed by the other ones by choosing for each pair of sentences above a model which satisfies these two, but not the third sentence — essentially, you are asked to find three binary relations, each satisfying just two of these properties.

* 2. Show the semantic entailment

$$\forall x\, P(x) \vee \forall x\, Q(x) \models \forall x\, (P(x) \vee Q(x)).$$

3. Prove $\forall x \neg\phi \models \neg\exists x\, \phi$; for that you have to take any model which satisfies $\forall x \neg\phi$ and you have to reason why this model must also satisfy $\neg\exists x\, \phi$. You should do this in a similar way to the examples in Section 2.4.2.
4. Let ϕ and ψ and η be formulas of predicate logic such that they contain no free variables.
   (a) If ψ is a semantically entailed by ϕ, is it necessarily the case that ψ is *not* semantically entailed by $\neg\phi$?
   * (b) If ψ is semantically entailed by $\phi \wedge \eta$, is it necessarily the case that ψ is semantically entailed by ϕ *and* semantically entailed by η?
   (c) If ψ is semantically entailed by ϕ or by η, is it necessarily the case that ψ is semantically entailed by $\phi \vee \eta$?
   (d) Explain why ψ is semantically entailed by ϕ iff $\phi \rightarrow \psi$ is valid, i.e. true in all models.
5. Show

$$\forall x\, (P(x) \vee Q(x)) \not\models \forall x\, P(x) \vee \forall x\, Q(x).$$

Thus, find a model which satisfies $\forall x\,(P(x) \vee Q(x))$, but not $\forall x\,P(x) \vee \forall x\,Q(x)$.

6. We call a set $\phi_1, \phi_2, \ldots, \phi_n$ of formulas *consistent* if there is a model of all predicate and function symbols involved such that all formulas $\phi_1, \phi_2, \ldots, \phi_n$ evaluate to T for that model. For each set of formulas below show that they are consistent:

   (a) $\forall x\,\neg S(x,x)$, $\exists x\,P(x)$, $\forall x\,\exists y\,S(x,y)$, $\forall x\,(P(x) \rightarrow \exists y\,S(y,x))$

   * (b) $\forall x\,\neg S(x,x)$, $\forall x\,\exists y\,S(x,y)$, $\forall x\,\forall y\,\forall z\,((S(x,y) \wedge S(y,z)) \rightarrow S(x,z))$

   (c) $(\forall x\,(P(x) \vee Q(x))) \rightarrow \exists y\,R(y)$, $\forall x\,(R(x) \rightarrow Q(x))$, $\exists y\,(\neg Q(y) \wedge P(y))$

   * (d) $\exists x\,S(x,x)$, $\forall x\,\forall y\,(S(x,y) \rightarrow (x = y))$.

7. Let P and Q be predicate symbols with one argument each. For the formulas below, check whether they are theorems. If not, then you have to find a model of P and Q such that the formula evaluates to F. Otherwise, you should find a proof using no premises:

   (a) $\forall x\,\forall y\,((P(x) \rightarrow P(y)) \wedge (P(y) \rightarrow P(x)))$

   (b) $(\forall x\,((P(x) \rightarrow Q(x)) \wedge (Q(x) \rightarrow P(x)))) \rightarrow ((\forall x\,P(x)) \rightarrow (\forall x\,Q(x)))$

   (c) $((\forall x\,P(x)) \rightarrow (\forall x\,Q(x))) \rightarrow (\forall x\,((P(x) \rightarrow Q(x)) \wedge (Q(x) \rightarrow P(x))))$

   (d) (Difficult.) $(\forall x\,\exists y\,(P(x) \rightarrow Q(y))) \rightarrow (\exists y\,\forall x\,(P(x) \rightarrow Q(y)))$.

8. For each of the formulas of predicate logic below, either find a model which does not satisfy it, or prove it without any premises:

   (a) $(\forall x\,\forall y\,(S(x,y) \rightarrow S(y,x))) \rightarrow (\forall x\,\neg S(x,x))$

   (b) $\exists y\,((\forall x\,P(x)) \rightarrow P(y))$

   (c) $(\forall x\,(P(x) \rightarrow \exists y\,Q(y))) \rightarrow (\forall x\,\exists y\,(P(x) \rightarrow Q(y)))$

   (d) $(\forall x\,\exists y\,(P(x) \rightarrow Q(y))) \rightarrow (\forall x\,(P(x) \rightarrow \exists y\,Q(y)))$

   (e) $\forall x\,\forall y\,(S(x,y) \rightarrow (\exists z\,(S(x,z) \wedge S(z,y))))$

   (f) $(\forall x\,\forall y\,(S(x,y) \rightarrow (x = y))) \rightarrow (\forall z\,\neg S(z,z))$

   (g) $(\forall x\,\exists y\,(S(x,y) \wedge ((S(x,y) \wedge S(y,x)) \rightarrow (x = y)))) \rightarrow (\neg\exists z\,\forall w\,(S(z,w)))$.

2.5 Undecidability of predicate logic

We conclude our introduction to predicate logic with some negative results. Given a formula ϕ in *propositional logic* we can, at least in principle, determine whether $\models \phi$ holds: if ϕ has n propositional atoms, then the truth

table of ϕ contains 2^n lines; and $\models \phi$ holds if, and only if, the column for ϕ contains only T entries.

The bad news is that such a mechanical procedure, working for all formulas ϕ, cannot be provided if ϕ is a formula in *predicate logic*. We will give a formal proof of this negative result, though we rely on an informal (yet intuitive) notion of computability.

The problem of determining whether a predicate logic formula is valid is known as a *decision problem*. A solution to a decision problem is a program (written in Java, C, or any other common language) that takes problem instances as input and *always* terminates, producing a correct 'yes' or 'no' output. In the case of the decision problem for predicate logic, the input to the program is an arbitrary formula ϕ of predicate logic and the program is correct if it produces 'yes' if the formula is valid and 'no' if it is not. Note that the program which solves a decision problem must terminate for all well-formed input: a program which goes on thinking about it for ever is not allowed.

The decision problem at hand is this:

Validity in predicate logic. *Given a logical formula ϕ in predicate logic, does $\models \phi$ hold, yes or no?*

We now show that this problem is not solvable; we cannot write a correct C or Java program that works for all ϕ. It is important to be clear about exactly what we are stating. Naturally, there are some ϕ which can easily be seen to be valid; and others which can easily be seen to be invalid. However, there are also some ϕ for which it is not easy. Every ϕ can, in principle, be discovered to be valid or not, if you are prepared to work arbitrarily hard at it; but there is no uniform mechanical procedure for determining whether ϕ is valid which will work for all ϕ.

We prove this by a well-known technique called *problem reduction*. That is, we take some other problem, of which we already know that it is not solvable, and we then show that the solvability of *our* problem entails the solvability of the other one. This is a beautiful application of the proof rule RAA, since we can then infer that our own problem cannot be solvable as well.

The problem that is known not to be solvable is interesting in its own right and, upon first reflection, does not seem to have a lot to do with predicate logic; it is the *Post correspondence problem*:

The Post correspondence problem. *Given a finite sequence of pairs* (s_1, t_1), $(s_2, t_2), \ldots, (s_k, t_k)$ *such that all* s_i *and* t_i *are binary strings of positive length, is there a sequence of indices* $i_1, i_2, \ldots, i_n$ *with* $n \geq 1$ *such that the concatenation of strings* $s_{i_1} s_{i_2} \ldots s_{i_n}$ *equals* $t_{i_1} t_{i_2} \ldots t_{i_n}$*?*

Here is an *instance* of the problem which we can solve successfully: the concrete correspondence problem instance C is given by a sequence of three pairs

$$C \stackrel{\text{def}}{=} ((1, 101), (10, 00), (011, 11))$$

so $s_1 \stackrel{\text{def}}{=} 1$, $s_2 \stackrel{\text{def}}{=} 10$ and $s_3 \stackrel{\text{def}}{=} 011$, whereas $t_1 \stackrel{\text{def}}{=} 101$, $t_2 \stackrel{\text{def}}{=} 00$ and $t_3 \stackrel{\text{def}}{=} 11$. A solution to the problem is the sequence of indices $(1, 3, 2, 3)$ since $s_1 s_3 s_2 s_3$ and $t_1 t_3 t_2 t_3$ both equal 101110011. Maybe you think that this problem must surely be solvable; but remember that a computational solution would have to be a program that solves *all* such problem instances. Things get a bit tougher already if we look at this (solvable) problem:

$$\begin{array}{cccc} s_1 \stackrel{\text{def}}{=} 001 & s_2 \stackrel{\text{def}}{=} 01 & s_3 \stackrel{\text{def}}{=} 01 & s_4 \stackrel{\text{def}}{=} 10 \\ t_1 \stackrel{\text{def}}{=} 0 & t_2 \stackrel{\text{def}}{=} 011 & t_3 \stackrel{\text{def}}{=} 101 & t_4 \stackrel{\text{def}}{=} 001. \end{array}$$

You are invited to solve this by hand, or by writing a program for this specific instance.

Note that the same number can occur in the sequence of indices, as happened in the first example in which 3 occurs twice. This means that the search space we are dealing with is infinite, which should give us a strong intuition that the problem is unsolvable. We do not formally prove it in this book.

The proof of the following theorem is due to the mathematician A. Church.

Theorem 2.17 *The decision problem of validity in predicate logic is undecidable: there is no procedure which, given any* ϕ*, decides whether* $\models \phi$ *holds.*

PROOF: As said before, we pretend that validity is decidable for predicate logic and thereby solve the (insoluble) Post correspondence problem. Given a correspondence problem instance C:

$$\begin{array}{cccc} s_1 & s_2 & \ldots & s_k \\ t_1 & t_2 & \ldots & t_k \end{array}$$

we need to be able to construct, within finite space and time, some formula ϕ of predicate logic such that $\models \phi$ holds iff the correspondence problem

instance C above has a solution. As function symbols, we choose a constant e and two function symbols f_0 and f_1 each of which requires one argument. We think of e as the empty string, or word, and f_0 and f_1 symbolically stand for concatenation with 0, respectively 1. So if $b_1 b_2 \ldots b_l$ is a binary string of bits, we can code that up as the term

$$f_{b_l}(f_{b_{l-1}} \ldots (f_{b_2}(f_{b_1}(e))) \ldots).$$

Note that this coding spells that word *backwards*. To facilitate reading those formulas, we abbreviate terms like $f_{b_l}(f_{b_{l-1}} \ldots (f_{b_2}(f_{b_1}(t))) \ldots)$ by $f_{b_1 b_2 \ldots b_l}(t)$.

We also require a predicate symbol P which expects two arguments. The intended meaning of $P(s,t)$ is that there is some sequence of indices $(i_1, i_2, \ldots, i_m)$ such that s is the term representing $s_{i_1} s_{i_2} \ldots s_{i_m}$ and t represents $t_{i_1} t_{i_2} \ldots t_{i_m}$. Thus, s constructs a string using the same sequence of indices as does t; only s uses the s_i whereas t uses the t_i.

Our sentence ϕ has the coarse structure

$$\phi_1 \wedge \phi_2 \rightarrow \phi_3$$

where we set

$$\begin{aligned}
\phi_1 &\stackrel{\text{def}}{=} \bigwedge_{i=1}^{k} P(f_{s_i}(e), f_{t_i}(e)) \\
\phi_2 &\stackrel{\text{def}}{=} \forall v \, \forall w \left(P(v,w) \rightarrow \bigwedge_{i=1}^{k} P(f_{s_i}(v), f_{t_i}(w)) \right) \\
\phi_3 &\stackrel{\text{def}}{=} \exists z \, P(z,z).
\end{aligned}$$

Our claim is that formula ϕ holds in all models iff the Post correspondence problem C has a solution.

First, let us assume that $\models \phi$. Our strategy is to find a model for ϕ which tells us there is a solution to the correspondence problem C simply by inspecting what it means for ϕ to satisfy that particular model. The universe of concrete values A of that model is the set of all finite, binary strings (including the empty one). The interpretation $e^{\mathcal{M}}$ of the constant e is just that empty word. The interpretation of f_0 is the unary function $f_0^{\mathcal{M}}$ which appends a 0 to a given word:

$$f_0^{\mathcal{M}}(s) \stackrel{\text{def}}{=} s0;$$

similarly,

$$f_1^{\mathcal{M}}(s) \stackrel{\text{def}}{=} s1$$

appends a 1 to a given word. The interpretation of P on $\mathcal{M}$ is just what we expect it to be:

$$P^{\mathcal{M}} \stackrel{\text{def}}{=} \{(s,t) \mid \text{there is a sequence of indices } (i_1, i_2, \ldots, i_m) \text{ such that } s \text{ equals } s_{i_1} s_{i_2} \ldots s_{i_m} \text{ and } t \text{ equals } t_{i_1} t_{i_2} \ldots t_{i_m}\},$$

where s and t are binary strings and the s_i and t_i are the data of the correspondence problem C. Thus, a pair of words (s,t) lies in $P^{\mathcal{M}}$ if, using the same sequence of indices $(i_1, i_2, \ldots, i_m)$, s is built using the corresponding s_i and t is built using the respective t_i.

We now show that the fact that ϕ holds in the model $\mathcal{M}$ implies that C is solvable. First, note that $\mathcal{M}$ satisfies ϕ_1 and ϕ_2. For example, ϕ_2 says about $\mathcal{M}$ that, if the pair (s,t) is in $P^{\mathcal{M}}$, then the pair $(s\,s_i, t\,t_i)$ is also in $P^{\mathcal{M}}$ for $i = 1,2,\ldots,k$ (you can verify this by inspecting the definition of $P^{\mathcal{M}}$). Now $(s,t) \in P^{\mathcal{M}}$ implies that there is some sequence $(i_1, i_2, \ldots, i_m)$ such that s equals $s_{i_1} s_{i_2} \ldots s_{i_m}$ and t equals $t_{i_1} t_{i_2} \ldots t_{i_m}$. We simply choose the new sequence $(i_1, i_2, \ldots, i_m, i)$ and observe that $s\,s_i$ equals $s_{i_1} s_{i_2} \ldots s_{i_m} s_i$ and $t\,t_i$ equals $t_{i_1} t_{i_2} \ldots t_{i_m} t_i$. (Why does $\mathcal{M} \models \phi_1$ hold?)

Since $\mathcal{M} \models \phi_1 \wedge \phi_2 \rightarrow \phi_3$ and $\mathcal{M} \models \phi_1 \wedge \phi_2$, it follows that $\mathcal{M} \models \phi_3$. By definition of ϕ_3 and $P^{\mathcal{M}}$, this tells us there is a solution to C.

Conversely, let us assume that the Post correspondence problem C has some solution, namely the sequence of indices $(i_1, i_2, \ldots, i_n)$. Now we have to show that, if $\mathcal{M}'$ is *any* model having a constant $e^{\mathcal{M}'}$, two unary functions, $f_0^{\mathcal{M}'}$ and $f_1^{\mathcal{M}'}$, and a binary predicate $P^{\mathcal{M}'}$, then that model has to satisfy ϕ. Notice that the root of the parse tree of ϕ is an implication, so this is the crucial clause for the definition of $\mathcal{M}' \models \phi$. By that very definition, we are already done if $\mathcal{M}' \not\models \phi_1$, or if $\mathcal{M}' \not\models \phi_2$. The harder part is therefore the one where $\mathcal{M}' \models \phi_1 \wedge \phi_2$, for in that case we need to verify $\mathcal{M}' \models \phi_3$ as well. The way we proceed here is by interpreting finite, binary strings in the domain of values A' of the model $\mathcal{M}'$. This is not unlike the coding of an interpreter for one programming language in another. The interpretation is done by a function interpret which is defined inductively on the data structure of finite, binary strings (we write ϵ for the empty word):

$$\begin{aligned} \mathsf{interpret}(\epsilon) &\stackrel{\text{def}}{=} e^{\mathcal{M}'} \\ \mathsf{interpret}(s0) &\stackrel{\text{def}}{=} f_0^{\mathcal{M}'}(\mathsf{interpret}(s)) \\ \mathsf{interpret}(s1) &\stackrel{\text{def}}{=} f_1^{\mathcal{M}'}(\mathsf{interpret}(s)). \end{aligned}$$

Note that interpret(s) is defined inductively on the length of s. This interpretation is, like the coding above, backwards; for example, the binary word

0100110 gets interpreted as

$$f_0^{\mathscr{M}'}(f_1^{\mathscr{M}'}(f_1^{\mathscr{M}'}(f_0^{\mathscr{M}'}(f_0^{\mathscr{M}'}(f_1^{\mathscr{M}'}(f_0^{\mathscr{M}'}(e^{\mathscr{M}'})))))))).$$

Note that

$$\mathsf{interpret}(b_1 b_2 \ldots b_l) = f_{b_l}^{\mathscr{M}'}(f_{b_{l-1}}^{\mathscr{M}'}(\ldots(f_{b_1}(e^{\mathscr{M}'})\ldots)))$$

is just the meaning of $f_s(e)$ in A', where $s \stackrel{\text{def}}{=} b_1 b_2 \ldots b_l$. Using that and the fact that $\mathscr{M}' \models \phi_1$, we conclude that $(\mathsf{interpret}(s_i), \mathsf{interpret}(t_i)) \in P^{\mathscr{M}'}$ for $i = 1, 2, \ldots, k$. Similarly, since $\mathscr{M}' \models \phi_2$, we know that for all $(s,t) \in P^{\mathscr{M}'}$ we have that $(\mathsf{interpret}(ss_i), \mathsf{interpret}(tt_i)) \in P^{\mathscr{M}'}$ for $i = 1, 2, \ldots, k$. Using these two facts, starting with $(s,t) = (s_{i_1}, t_{i_1})$, we repeatedly use the latter observation to obtain

$$(\mathsf{interpret}(s_{i_1} s_{i_2} \ldots s_{i_n}), \mathsf{interpret}(t_{i_1} t_{i_2} \ldots t_{i_n})) \in P^{\mathscr{M}'}.$$

Since $s_{i_1} s_{i_2} \ldots s_{i_n}$ and $t_{i_1} t_{i_2} \ldots t_{i_n}$ together form a solution of C, they are equal; and therefore the elements $\mathsf{interpret}(s_{i_1} s_{i_2} \ldots s_{i_n})$ and $\mathsf{interpret}(t_{i_1} t_{i_2} \ldots t_{i_n})$ are the same in A', for interpreting the same thing gets you the same result. Hence the pair

$$(\mathsf{interpret}(s_{i_1} s_{i_2} \ldots s_{i_n}), \mathsf{interpret}(t_{i_1} t_{i_2} \ldots t_{i_n})) \in P^{\mathscr{M}'}$$

verifies $\exists z\, P(z,z)$ in $\mathscr{M}'$ and thus $\mathscr{M}' \models \phi_3$. □

There are two more negative results which we now get quite easily. Let us say that a formula ϕ is *satisfiable* if there is some model $\mathscr{M}$ such that $\mathscr{M} \models \phi$. This property is not to be taken for granted; the formula

$$\exists x\,(P(x) \wedge \neg P(x))$$

is clearly unsatisfiable. More interesting is the observation that ϕ is unsatisfiable if, and only if, $\neg\phi$ is valid, i.e. holds in *all* models. This is an immediate consequence of the definitional clause $\mathscr{M} \models \neg\phi$ for negation. Since we can't compute validity, it follows that we cannot compute satisfiability either.

The other undecidability result comes from the soundness and completeness of predicate logic:

$$\models \phi \quad \text{iff} \quad \vdash \phi \tag{2.2}$$

which we do not prove in this text. Since we can't decide validity, we cannot decide *provability* either, on the basis of (2.2). One might reflect on that last negative result a bit. It means bad news if one wants to implement

perfect theorem provers which can mechanically produce a proof of a given formula, or refute it. It means good news, though, if we like the thought that machines still need a little bit of human help. Creativity seems to have limits if we leave it to machines alone.

EXERCISES 2.8

1. Assuming that our proof calculus for predicate logic is sound (see exercise 2), show that the following sequents cannot be proved by finding for each sequent a model such that all formulas to the left of $\vdash$ evaluate to T and the sole formula to the right of $\vdash$ evaluates to F (explain why this guarantees the non-existence of a proof):

   (a) $\forall x\,(P(x) \vee Q(x)) \vdash \forall x\,P(x) \vee \forall x\,Q(x)$

   * (b) $\forall x\,(P(x) \rightarrow R(x)),\ \forall x\,(Q(x) \rightarrow R(x)) \vdash \exists x\,(P(x) \wedge Q(x))$

   (c) $(\forall x\,P(x)) \rightarrow L \vdash \forall x\,(P(x) \rightarrow L)$, where L is a predicate symbol with no arguments

   * (d) $\forall x\,\exists y\,S(x, y) \vdash \exists y\,\forall x\,S(x, y)$

   (e) $\exists x\,P(x),\ \exists y\,Q(y) \vdash \exists z\,(P(z) \wedge Q(z))$.

2. To show the soundness of our natural deduction rules for predicate logic, it intuitively suffices to show that the conclusion of a proof rule is true provided that all its premises are true. What additional complication arises due to the presence of variables and quantifiers? Can you precisely formalise the necessary induction hypothesis for proving soundness?

* 3. Assuming that our proof calculus for predicate logic is sound (see exercise 2), show that the following two sequents cannot be proved in predicate logic. Relying on the soundness of our proof calculus, it suffices to do the following: for each sequent you need to specify a model such that the formula on the left of $\vdash$ holds whereas the one to the right of $\vdash$ doesn't.

   (a) $\exists x\,(\neg P(x) \wedge Q(x)) \vdash \forall x\,(P(x) \rightarrow Q(x))$

   (b) $\exists x\,(\neg P(x) \vee \neg Q(x)) \vdash \forall x\,(P(x) \vee Q(x))$.

2.6 Bibliographic notes

Many design decisions have been taken in the development of predicate logic in the form known today. The Greeks and the medievals had systems in which many of the examples and exercises in this book could be represented, but

nothing that we would recognise as predicate logic emerged until the work of Gottlob Frege in 1879, printed in [Fre03]. An account of the contributions of the many other people involved in the development of logic can be found in the first few pages of W. Hodges' chapter in [Hod83].

There are many books covering classical logic and its use in computer science; we give a few incomplete pointers to the literature. The books [SA91], [vD89] and [Gal87] cover more theoretical applications than those in this book, including type theory, logic programming, algebraic specification and term-rewriting systems. An approach focusing on automatic theorem proving is taken by [Fit96]. Books which study the mathematical aspects of predicate logic in greater detail, such as completeness of the proof systems and incompleteness of first-order arithmetic, include [Ham78] and [Hod83].

Most of these books present other proof systems besides natural deduction such as axiomatic systems and tableau systems. Although natural deduction has the advantages of elegance and simplicity over axiomatic methods, there are few expositions of it in logic books aimed at a computer science audience. One exception to this is the book [BEKV94], which is the first one to present the rules for quantifiers in the form we used here. A natural deduction theorem prover called Jape has been developed, in which one can vary the set of available rules and specify new ones[1].

A standard reference for computability theory is [BJ80]. A proof for the undecidability of the Post correspondence problem can be found in the text book [Tay98].

The second instance of a Post correspondence problem is taken from [Sch92]. A text on the fundamentals of databases systems is [EN94].

[1] www.comlab.ox.ac.uk/oucl/users/bernard.sufrin/jape.shtml