Chapter 4

The World According to Predicate Logic

Overview At this stage of our course, you already know propositional logic, the system for reasoning with sentence combination, which forms the basic top-level structure of argumentation. Then we zoomed in further on actual natural language forms, and saw how sentences make quantified statements about properties of objects, providing a classification of the world in terms of a hierarchy of smaller or larger predicates. You also learnt the basics of syllogistic reasoning with such hierarchies.

In this Chapter, we look still more deeply into what we can actually say about the world. You are going to learn the full system of 'predicate logic' of objects, their properties, but also the relations between them, and about these, arbitrary forms of quantification. This is the most important system in logic today, because it is a *universal language* for talking about *structure*. A structure is any situation with objects, properties and relations, and it can be anything from daily life to science: your family tree, the information about you and your friends on Facebook, the design of the town you live in, but also the structure of the number systems that are used in mathematics, geometrical spaces, or the universe of sets. In the examples for this chapter, we will remind you constantly of this broad range from science to daily life.

Predicate logic has been used to increase precision in describing and studying structures from linguistics and philosophy to mathematics and computer science. Being able to use it is a basic skill in many different research communities, and you can find its notation in many scientific publications. In fact, it has even served as a model for designing new computer languages, as you will see in one of our Outlooks. In this chapter, you will learn how predicate logic works, first informally with many examples, later with more formal definitions, and eventually, with outlooks showing you how this system sits at the interface of many disciplines. But this power comes at a price. This chapter is not easy, and mastering predicate logic until it comes naturally to you takes a while – as successive generations of students (including your teachers) have found.

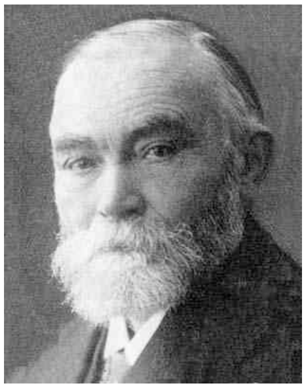
4.1 Learning the Language by Doing

Zooming in on the world Propositional logic classifies situations in terms of 'not', 'and', 'or' combinations of basic propositions. This truth-table perspective is powerful in its own way (it is the basis of all the digital circuits running your computer as you are reading this), but poor in other respects. Basic propositions in propositional logic are not assumed to have internal structure. "John walks" is translated as p, "John talks" as q, and the information that both statements are about John gets lost. Predicate logic looks at the internal structure of such basic facts. It translates "John walks" as Wj and "John talks" as Tj, making it clear that the two facts express two properties of the same person, named by the constant j.

As we said, predicate logic can talk about the internal structure of situations, especially, the objects that occur, properties of these objects, but also their relations to each other. In addition, predicate logic has a powerful analysis of universal quantification (all, every, each, $\ldots$) and existential quantification (some, a, $\ldots$). This brings it much closer to two languages that you already knew before this course: the natural languages in the common sense world of our daily activities, and the symbolic languages of mathematics and the sciences. Predicate logic is a bit of both, though in decisive points, it differs from natural language and follows a more mathematical system. That is precisely why you are learning something new in this chapter: an additional style of thinking.

Two founding fathers Predicate logic is a streamlined version of a "language of thought" that was proposed in 1878 by the German philosopher and mathematician Gottlob Frege (1848 – 1925). The experience of a century of work with this language is that, in principle, it can write all of mathematics as we know it today. Around the same time, essentially the same language was discovered by the American philosopher and logician Charles Saunders Peirce. Peirce's interest was general reasoning in science and daily life, and his ideas are still inspirational to modern areas philosophers, semioticists, and researchers in Artificial Intelligence. Together, these two pioneers stand for the full range of predicate logic.

Charles Sanders Peirce


Gottlob Frege

We will now introduce predicate logic via a sequence of examples. Grammar comes later: further on in this chapter we give precise grammatical definitions, plus other information.

> *If you are more technically wired, you can skim the next four introductory sections, and then go straight to the formal part of this chapter.*

We do not start in a vacuum here: the natural language that you know already is a running source of examples and, in some cases, contrasts:

The basic vocabulary We first need names for *objects*. We use constants ('proper names') a, b, c, $\ldots$ for special objects, and variables x, y, z, $\ldots$ when the object is indefinite. Later on, we will also talk about function symbols for complex objects.

Then, we need to talk about *properties and predicates* of objects. Capital letters are predicate letters, with different numbers of 'arguments' (i.e., the objects they relate) indicated. In natural language, 1-place predicates are intransitive verbs ("walk") and common nouns ("boy"), 2-place predicates are transitive verbs ("see"), and 3-place predicates are so-called ditransitive verbs ("give"). 1-place predicates are also called *unary predicates*, 2-place predicates are called *binary predicates*, and 3-place predicates are called *ternary predicates*. In natural language ternary predicates are enough to express the most complex verb pattern you can get, but logical languages can handle any number of arguments.

Next, there is still *sentence combination*. Predicate logic gratefully incorporates the usual operations from propositional logic: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$. But in addition, and very importantly, it has a powerful way of expressing *quantification*. Predicate logic has quantifiers $\forall x$ ("for all x") and $\exists x$ ("there exists an x") tagged by *variables* for objects, that can express an amazing number of things, as you will soon see.

From natural language to predicate logic For now, here is a long list of examples showing you the underlying 'logical form' of the statements that you would normally make when speaking or writing. Along the way we will point out various important features.

Atomic statements We start with the simplest statements about objects:

natural language	*logical formula*
John walks	Wj
John is a boy	Bj
He walks	Wx
John sees Mary	Sjm
John gives Mary the book	$Gjmb$

Predicate logic treats both verbs and nouns as standing for properties of objects, even though their syntax and communicative function is different in natural language. The predicate logical form of “John walks” uses a predicate letter and a single constant. The form of “John is a boy” also uses a predicate letter with a constant: Bj.

These examples demonstrate the variety of predication in natural language: intransitive verbs like ‘Walk” take one object, transitive verbs like “see” take two, verbs like “give” even take three. The same variety occurs in mathematics as we will see a bit later, and it is essential to predicate logic: atomic statements express basic properties of one or more objects together. In the history of logic, this is a relatively late insight. The theory of syllogistics describes only properties of single objects, not relations between two or more objects.

Exercise 4.1 The hold of the syllogistic of our preceding chapter, and its emphasis on “unary” properties of single objects has been so strong that many people have tried to reduce binary predicates to unary ones. One frequent proposal has been to read, say, “x is smaller than y” as “x is small and y is not small”. Discuss this, and show why it is not adequate. Does it help here to make the property “small” context-dependent: “small compared to...”?

Translation key Note that in writing predicate logical translations, one has to choose a “key” that matches natural language expressions with corresponding logical letters. And then stick to it. For mnemonic purposes, we often choose a capital letter for a predicate as close to the natural language expression as we can (e.g., B for “boy”). Technically, in the logical notation, we should indicate the exact number of object places that the predicate takes (“B has one object place”), but we drop this information when it is clear from context. The object places of predicates are also called *argument places*. If a predicate takes more than one argument, the key should say in which order you read the arguments. E.g., our key here is that Sjm says that John sees Mary, not that Mary sees John. The latter would be Smj.

Predicates in language and mathematics Let us discuss predicates a bit further, since their variety is so important to predicate logic. In mathematics, 2-place predicates are most frequent. Common examples are $=$ (‘is equal to’), $<$ (‘is smaller than’), $\in$ (‘is an element of’). It is usual to write these predicates in between their arguments: $2 < 3$. (We will say more about the expressive possibilities of the predicate “$=$” on page 4-41.) Occasionally, we also have 3-place predicates. An example from geometry is “x lies between y and z”, an example from natural language is the word “give” (with a giver, an object, and a recipient).

informal mathematics	*logical/mathematical formula*
Two is smaller than three	$2 < 3$
x is smaller than three	$x < 3$
x is even (i.e., 2 divides x)	$2\|x$
Point p lies between q and r	$Bpqr$

In the special case of talking about mathematics there are standard names for objects and relations. In $x < 3$, the term "3" is a constant that names a particular natural number, and "<" is a standard name for a specific relation. The notation $x|y$ expresses that x is a divisor of y, i.e., that division of y by x leaves no remainder. Natural language also has special names for distinguished objects, such as "Alexander the Great", "Indira Ghandi", or "The Great Wall".

Note that betweenness is not a conjunction of "x lies between y" and "x lies between z": that would be nonsense. However, an abbreviation that is often used in mathematics is $x < y < z$, to express that the number y is in between x and z. This is not an example of real 3-place predicate. Rather, it is an abbreviation of $x < y \wedge y < z$. In this special case, when an order is 'linear', betweenness does reduce to a conjunction after all.

Exercise 4.2 Express $\neg(x < y < z)$ in terms of the binary predicate $<$ and propositional connectives, using the fact that $x < y < z$ is an abbreviation of $x < y \wedge y < z$.

The standard in predicate logic is to write the predicate first, then the objects. The exceptions to this rule are the names for binary relations in mathematics: $<$ for less than, $>$ for greater than, and so on. The general rule is for uniformity, and it takes getting used to. Many natural languages put predicates in the middle (English, French, but also the informal language of mathematics), but other languages put them first, or last. Dutch and German are interesting, since they put predicates in the middle in main clauses ("Jan zag Marie"), but shift the predicate to the end in subordinate clauses ("Ik hoorde dat Jan Marie zag").

Referring to objects: pronouns and variables We already saw how proper names like "John" or "Mary" refer to specific objects, for which we wrote constants like a, b. But both natural language and mathematics use 'variable names' as well, that stand for different objects in different contexts. Pronouns in language work like this: "John sees her" (Sjx) refers to some contextually determined female "her", and $x < 2$ expresses that some contextually determined number x is smaller than 2. Think of a geometry textbook where x is introduced as the side of a triangle with two other sides of length 1. The famous author Italo Calvino once wittily called pronouns "the lice of thought" [Cal88]. But are they just a nuisance? To the contrary, what pronouns do is provide coherence in what you say, by referring back to the same individual in the right places. That is also exactly what mathematical variables do. So we get analogies between pronouns in natural language and contextually determined variables in mathematics, like:

John sees her	Sjx
He sees her	Syx
This is less than that	$x < y$
He sees himself	Sxx

Note that whether $x < y$ is true totally depends on specifying x and y. It is true for $x = 2$ and $y = 5$ but false for $x = 5$ and $y = 2$. 'This' and 'that' in natural language are demonstrative pronouns for pointing at things. Mathematicians use variables as pointers. Instead of "suppose this is some number greater than one" they say "suppose x is a number greater than one". Next, they use x to refer to this number.

Adding on propositional logic Propositional operators can be added in the obvious way to the preceding statements, with the same function as before:

John does not see Mary	$\neg Sjm$
Three is not less than two	$\neg 3 < 2$, or abbreviated: $3 \not< 2$.
John sees Mary or Paula	$Sjm \vee Sjp$
Three is less than three or three is less than four	$3 < 3 \vee 3 < 4$.
x is odd (i.e., two does not divide x)	$\neg(2 \vert x)$
If John sees Mary, he is happy	$Sjm \rightarrow Hj$

As a small detail in style, in the last example, natural language uses a pronoun ("he is happy"), while the logical translation does not use a variable but a second occurrence of the constant j to refer to the same object as before. Logic is more precise, natural language is more flexible. The reuse of the constant j rules out any possibility of misunderstanding, while the correct interpretation of "he" still depends on context.

Exercise 4.3 Translate the following sentences into predicate logical formulas:

(1) If John loves Mary, then Mary loves John too.

(2) John and Mary love each other.

(3) John and Mary don't love each other.

(4) If John and Peter love Mary then neither Peter nor Mary loves John.

Exercise 4.4 Rephrase "$x \leq y \wedge y \leq z$" in predicate logic, using binary relations $<$ for "less than" and $=$ for "equal".

Exercise 4.5 Rephrase "$\neg(x \leq y \wedge y \leq z)$" in predicate logic, using binary relations $<$ for "less than" and $=$ for "equal".

A first glimpse of quantifiers Now we move to the most important new operators, quantifiers which say that objects exist without naming them explicitly:

Someone walks	$\exists x W x$
Some boy walks	$\exists x(Bx \wedge Wx)$
A boy walks	$\exists x(Bx \wedge Wx)$
John sees a girl	$\exists x(Gx \wedge Sjx)$
A girl sees John	$\exists x(Gx \wedge Sxj)$
A girl sees herself	$\exists x(Gx \wedge Sxx)$

Here you can see what variables do: they keep track of which object does what when that object occurs several times in the sentence. With long sentences or long texts, a systematic mechanism like this becomes crucial in keeping reasoning straight.

Here are some examples with universal quantifiers:

Everyone walks	$\forall x W x$
Every boy walks	$\forall x(Bx \to Wx)$
Every girl sees Mary	$\forall x(Gx \to Sxm)$

Note that "Some boy walks" is translated with a conjunction symbol, and "Every boy walks" is translated using an implication symbol. The following exercise explains why.

Exercise 4.6 Let B be the predicate for "boy" and W the predicate for "walk".

(1) What does $\forall x(Bx \wedge Wx)$ express?

(2) And what does $\exists x(Bx \to Wx)$ express?

At first, you may find this predicate-logical formulation somewhat strange. But give it some more thought, and you may come to see that the predicate-logical formulas really get to the heart of the meaning and structure of objects.

Remark on natural language and logic Our examples show similarities between natural language and logic, but also a bit of friction. It may seem that "John is a boy" contains an existential quantifier ("there is a boy"), but this appearance is misleading. This has been much discussed: already ancient Greek logicians felt that their own language has a redundancy here: the indefinite article "a" is redundant. Indeed, many natural languages do not put an "a" in this position. An example from Latin: *"puer est"* ("he is a child"). But such languages also do not put an "a" in positions where there is a real existential quantifier. Another example from Latin: *"puer natus est"* ("a child is born"). Likewise, the ancient Greek logicians already observed that the "is" of predication in "John is a boy" does not seem to do any real work, and again, might just be an accident of language. As one can imagine, debates about how natural language structure relates to logical form are far from over — and they remain a powerful inspiration for new research.

Quantifier combinations The advantages of the austere notation of predicate logic quickly become clearer when we combine quantifiers. This creates patterns that are essentially more sophisticated than those in our previous chapter on syllogistic reasoning. Combined patterns occur widely in science and natural language, but predicate logic makes them especially perspicuous:

Everyone sees someone	$\forall x \exists y Sxy$
Someone sees everyone	$\exists x \forall y Sxy$
Everyone is seen by someone	$\forall x \exists y Syx$
Someone is seen by everyone	$\exists x \forall y Syx$

You can again think of mathematical examples just as easily:“every number has a greater number”, “some number has no smaller number”, “some sets have no elements”, etc.

The domain of quantifiers Quantifiers in natural and formal languages come with an additional feature, that we left implicit so far. The quantifiers range over all objects in some given set of relevant objects, the *domain of discourse*. In the above natural language examples, the domain only contains human beings, perhaps even just a small set of them — in mathematical examples, it can be numbers or geometrical figures. But in principle, every set of objects, large or small, can be a domain of discourse. We will make this feature more explicit later when we talk about the semantics of predicate logic.

Technically, the restriction to domains of discourse is sometimes reflected in how we “restrict” a quantifier to some relevant subset, indicated by a unary predicate. For instance, in the obvious translations of syllogistic statements like “All A are B”, the formula $\forall x(\mathbf{A}x \rightarrow Bx)$ has its universal quantifier restricted to the predicate A, and likewise, the existential quantifier for “Some A are B” is restricted as follows: $\exists x(\mathbf{A}x \wedge Bx)$.

This completes our first tour of predicate logic. The next section will help you in much more detail with finding logical forms for natural language sentences.

4.2 Practising Translations

In a logic course, going from sentences to formulas is taught as a sort of art allowing you to see the structure of ordinary assertions, and doing inference with them. And this art also makes sense for our other extreme of mathematics: mathematicians also speak natural language, be it with many special notations, and they have never switched completely to using only formulas. The modern area of “natural language processing” has developed the translation process also into a sort of science, where computers actually translate given natural language sentences into logical representations. In what follows, we go through some detailed examples that may help you develop a methodical style of translating.

If you feel that you get the picture by now, you can skip this section.

Translating simple syllogistic sentences Here is how the syllogistic statements of the preceding chapter can be expressed in predicate logic:

All A are B	$\forall x(Ax \to Bx)$
No A are B	$\neg\exists x(Ax \land Bx)$
Some A are B	$\exists x(Ax \land Bx)$
Not all A are B	$\neg\forall x(Ax \to Bx)$

Exercise 4.7 Give the predicate logical formulas for the following syllogistic statements:

(1) No B are C.

(2) Some A are C.

But the real power of predicate logic shows only in combinations.

Translating sentences with multiple quantifiers We can also use these patterns for finding translations for sentences involving double quantifications, as in the following example:

$$\text{Every boy loves a girl.} \tag{4.1}$$

To begin with we translate the pattern for "All B are …":

$$\forall x\,(Bx \to \varphi(x)) \tag{4.2}$$

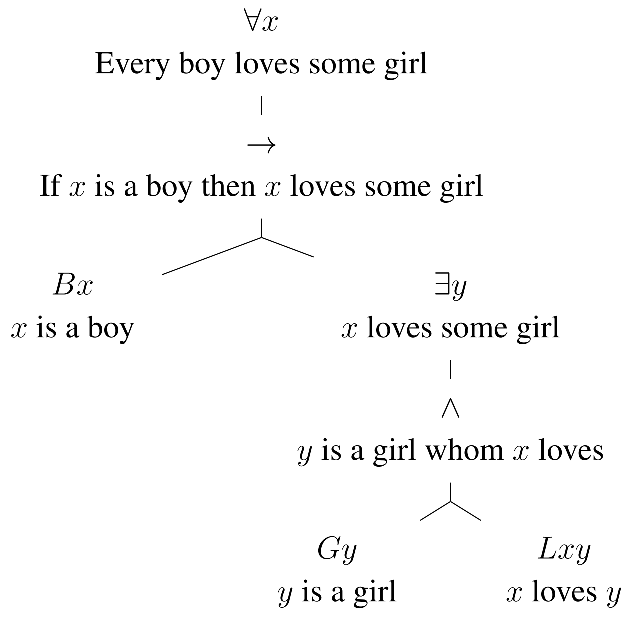
Here B is a unary predicate to represent 'boy', and $\varphi(x)$ stands for the property that we want to assign to all the boys x: "x loves a girl". This part of the sentence is in fact something of the form "Some C are D", where C represents the class of girls and D the class of those objects which are loved by x. The predicate logical translation of $\varphi(x)$ therefore looks as follows:

$$\exists y\,(Gy \land Lxy) \tag{4.3}$$

with Gy for "y is a girl" and Lxy for "x loves y". Substitution of this translation for $\varphi(x)$ in (4.2) gives us the full predicate logical translation of (4.1):

$$\forall x\,(Bx \to \exists y\,(Gy \land Lxy)) \tag{4.4}$$

In the following figure, a parse tree of the translation is given which shows the compositional structure of predicate logic. Every subformula corresponds to a sentence in natural language:



(4.5)

This way of constructing translations can be of great help in finding predicate logical formulas for natural language sentences.

In ordinary language we are used to writing from left to right, but in predicate logic the order of putting together connectives and quantifiers is often non-linear. The following longer example illustrates this.

No girl who loves a boy is not loved by some boy. (4.6)

Although this sentence looks quite complicated, its surface structure coincides with the syllogistic form “No A are B”, and so our first translation step is:

$$\neg\exists x\,(\varphi(x) \wedge \psi(x)) \tag{4.7}$$

where $\varphi(x)$ are those x who are girls who love some boy, and $\psi(x)$ represents the class of those x who are loved by no boy. The first part, $\varphi(x)$, is a conjunction of two properties: x is a girl *and* x loves a boy. This is just a small step towards a complete translation:

$$\neg\exists x\,((Gx \wedge \varphi_1(x)) \wedge \psi(x)) \tag{4.8}$$

with $\varphi_1(x)$ representing “x loves a boy”. This part can be translated into:

$$\exists y\,(By \wedge Lxy) \tag{4.9}$$

Substitution of this result for $\varphi_1(x)$ in (4.8) gives us the following intermediate result:

$$\neg\exists x\,((Gx \wedge \exists y\,(By \wedge Lxy)) \wedge \psi(x)) \tag{4.10}$$

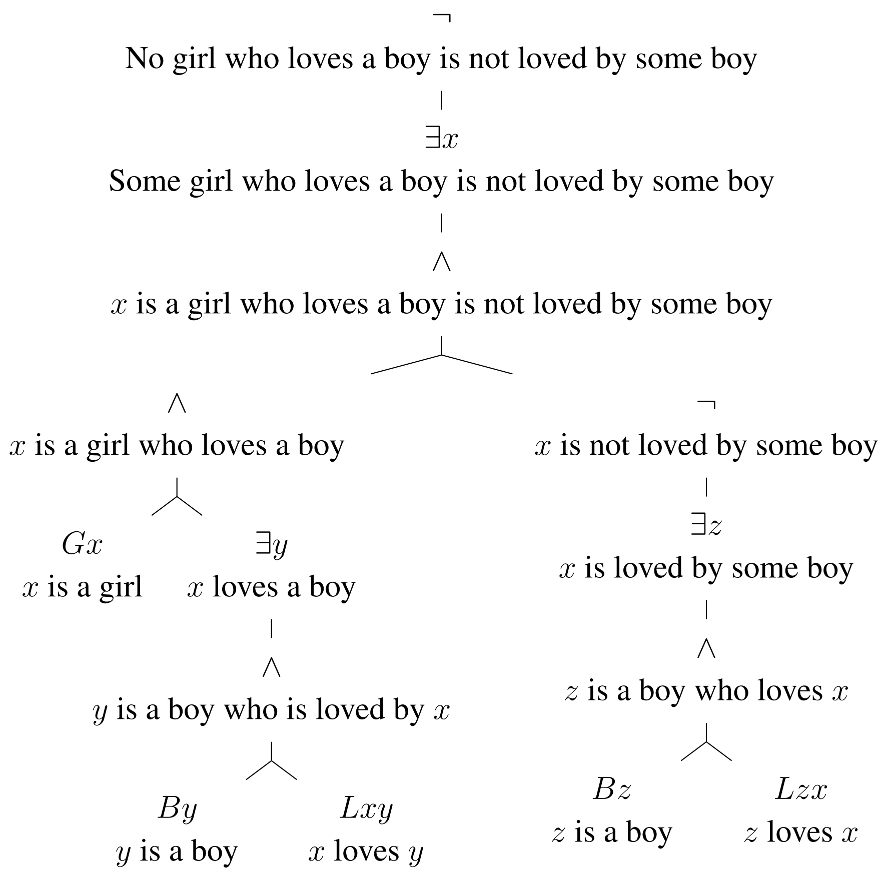
The subformula $\psi(x)$ represents "No boy loves x", which can be translated into:

$$\neg\exists z\,(Bz \wedge Lzx) \tag{4.11}$$

This yields the final result:

$$\neg\exists x\,((Gx \wedge \exists y\,(By \wedge Lxy)) \wedge \neg\exists z\,(Bz \wedge Lzx)) \tag{4.12}$$

Below a complete composition tree is given for this translation. Again, every subformula can be paraphrased in natural language:


$\neg$
No girl who loves a boy is not loved by some boy
$\exists x$
Some girl who loves a boy is not loved by some boy
$\wedge$
x is a girl who loves a boy is not loved by some boy
$\wedge$
x is a girl who loves a boy
$\neg$
x is not loved by some boy
Gx
x is a girl
$\exists y$
x loves a boy
$\exists z$
x is loved by some boy
$\wedge$
y is a boy who is loved by x
$\wedge$
z is a boy who loves x
By
y is a boy
Lxy
x loves y
Bz
z is a boy
Lzx
z loves x

(4.13)

Iterating quantifiers: twice, and more Two-quantifier combinations occur in natural language as we have just seen, and they are also very common in mathematics. The same logical form that expressed 'Everyone sees someone' is also that for a statement like 'Every number has a larger number'. And the above form for 'Some girl sees every boy' is also that for 'There is an odd number that divides every even number' (namely, the number 1).

Can there be still higher nestings of quantifiers? Yes, indeed. For instance, three quantifiers are involved in the famous saying that "You can fool some people some of the time, and you can fool some people all of the time, but you cannot fool all people all of the time". To see that this really involves three quantifiers, observe that the "you" must be read as "someone". The translation of "Someone can fool some people some of the time" (with P for "person", T for "instant of time", F for "fooling"):

$$\exists x(Px \wedge \exists y(Py \wedge \exists z(Tz \wedge Fxyz))).$$

And for "Someone cannot fool all people all of the time":

$$\neg\exists x(Px \wedge \forall y(Py \rightarrow \forall z(Tz \rightarrow Fxyz))).$$

Likewise, three-quantifier combinations occur in mathematics. A typical example is the definition of 'continuity' of a function f in a point x:

> For every number r, there is a number s such that for all y with $|x - y| < s$: $|f(x) - f(y)| < r$.

Nestings of four quantifiers are rare, they get hard for humans to understand.

Exercise 4.8 Assume the domain of discourse to be all human beings. Translate the following sentences into predicate logic:

(1) Augustus is not loved by everyone. (Use a for Augustus, L for love.)

(2) Augustus and Livia respect each other. (Use a for Augustus, l for Livia and R for respect.)

(3) Livia respects everyone who loves Augustus.

Exercise 4.9 Assume the domain of discourse is all animals. Translate: *Some birds do not fly.* (Use B for being a bird and F for being able to fly.)

Exercise 4.10 For each of the following, specify an appropriate domain of discourse, specify a key, and translate into predicate logic. (Note: you have to understand what a sentence means before you can attempt to translate it.)

(1) Dogs that bark do not bite.

(2) All that glitters is not gold.

(3) Friends of Michelle's friends are her friends.

(4) There is a least natural number.

(5) There is no largest prime number.

Exercise 4.11 Translate the following sentences into predicate logical formulas. Assume the domain of discourse is human beings.

(1) Every boy loves Mary.

(2) Not all girls love themselves.

(3) No boy or girl loves Peter.

(4) Peter loves some girl that loves John.

Exercise 4.12 For each sentence from the previous exercise, draw a picture that makes it true.

Exercise 4.13 Translate the following sentences into predicate logical formulas. Assume the domain of discourse is human beings.

(1) Some boy doesn't love all girls.

(2) Every boy who loves a girl is also loved by some girl.

(3) Every girl who loves all boys does not love every girl.

(4) No girl who does not love a boy loves a girl who loves a boy.

4.3 Reasoning Patterns with Quantifiers

Let us now turn to the kind of reasoning that predicate logic brings to light. As with earlier systems, the basis here is your intuitive understanding of reasoning, in this case, with quantifiers. The formal system then makes this more precise and systematic. In this section, we look at some valid inferences, without formal analysis: we are appealing to your intuitions.

Monadic Predicate Logic Before unleashing the full power of quantifiers in predicate logic, we first consider a more modest stepping stone.

The language of the Syllogism (Chapter 3) is a small fragment of predicate logic that imposes a restriction on the form of the predicates that are allowed: they have to be "unary properties", with atomic statements involving one object only. This special system with only 1-place predicates (unary properties of objects) is called *monadic predicate logic*. This restriction on predicate form curbs the power of quantification considerably, but it also makes it easier to understand. Let us see how syllogistic reasoning can be expressed in monadic predicate logic.

Consider the syllogism with premises "All A are B" and "All B are C" and conclusion 'All A are C". It corresponds to the valid predicate-logical inference

$$\forall x(Ax \to Bx), \forall x(Bx \to Cx) \textit{ imply } \forall x(Cx \to Ax)$$

by using a small variation of one of the methods of Chapter 3.

Of course, you have already learnt the Venn Diagram method that tests such inferences for validity or invalidity. More in terms of predicate logic, here are some further patterns. Syllogistic theory has the following equivalences:

- *Not all A are B* has the same meaning as *Some A are not B*.
- *All A are not B* has the same meaning is *there are no A that are also B*.

The predicate logical versions of these equivalences give important information about the interaction between quantification and negation:

- $\neg\forall x(Ax \to Bx)$ is equivalent to $\exists x\neg(Ax \to Bx)$, which is in turn equivalent to $\exists x(Ax \land \neg Bx)$,
- $\forall x(Ax \to \neg Bx)$ is equivalent to $\neg\exists x\neg(Ax \to \neg Bx)$, which is in turn equivalent to $\neg\exists x(Ax \land Bx)$.

From this we can distill some important general quantification principles:

- $\neg\forall x\varphi$ is equivalent to $\exists x\neg\varphi$,
- $\neg\exists x\varphi$ is equivalent to $\forall x\neg\varphi$.

Reflecting on these principles a bit further, we see that negation allows us to express one quantifier in terms of the other, as follows:

- $\forall x\varphi$ is equivalent to $\neg\exists x\neg\varphi$,
- $\exists x\varphi$ is equivalent to $\neg\forall x\neg\varphi$.

Exercise 4.14 Translate the following syllogistic statements into predicate logic, without using existential quantifiers:

(1) Some A are B.

(2) Some A are not B.

(3) No A are B.

Exercise 4.15 Translate the following syllogistic statements into predicate logic, without using universal quantifiers:

(1) All A are B.

(2) All A are non-B.

(3) No A are B.

Actually, monadic predicate logic also has basic inferences that are not syllogistic in nature. Here is a key example:

> $\forall$ distributes over $\wedge$.

On the other hand, $\forall$ does not distribute over $\vee$. Here is a simple counterexample. The Greeks held the view that all people are either Greeks or Barbarians. So, with *all people* as the domain of discourse:

$$\forall x(Gx \vee Bx).$$

But they did certainly not hold the view that either all people are Greek or all people are Barbarians. For they knew that the following was *false*:

$$\forall x Gx \vee \forall x Bx.$$

Exercise 4.16 What are the analogous observations for $\exists$, $\wedge$ and $\vee$?

Syllogistic reasoning is still very simple, and indeed, validity in monadic predicate logic, which has unary predicates only, is still *decidable*: there are mechanical methods for testing whether given inferences are valid, as we have seen in Chapter 3. This is no longer true for predicate logic as a whole, but we will come to that point only later.

Logic and natural language once more Modern logicians view the syllogistic as a tiny (monadic) fragment of the much richer full natural language. From that point of view, traditional logic was extremely weak, and its success in history becomes hard to understand. But this is an essential misunderstanding of how a logical system functions. Syllogistic forms are used for *analysis* of given sentences, and they provide a sort of top-level logical form, while the predicates A, B, etc. may stand for large chunks of natural language text of potentially very high (non-syllogistic!) complexity. For instance, here is a surface "All A are B" form: "All prisoners who have inflicted much damage on several people (A) fall under the provisions of most criminal codes invented by different cultures (B)".

In this sense you should not misunderstand the art of translation that we have shown you. Outside the realm of our toy examples, it is usually hopeless to translate a complete natural language sentence into a logical formula. The point is rather that you formalize part of the outer structure of the sentences involved, say, enough to see whether a claimed inference is valid or not. Leaving large internal chunks of the sentences unformalized is not a problem in such a case: it is rather the sign of a certain sense for taste and elegance.

Reasoning with binary and higher predicates, and with quantifiers Valid reasoning in predicate logic extends far beyond syllogistic forms. The following is a famous 19th century example of a non-syllogistic valid inference involving binary relations:

> "All horses are animal", therefore: "All horse tails are animal tails"

Using a binary predicate P for "possess" or "have", we can translate this as follows:

$$\frac{\forall x(Hx \to Ax)}{\forall y((Ty \land \exists x(Hx \land Pxy)) \to \exists z(Az \land Pzy))}$$

This is one instance of a general phenomenon: "monotonicity inferences" with predicate replacement of the sort discussed in the preceding chapter work in much greater generality than just the syllogistic. Predicate logic tells you how.

Another natural set of inferences has to do with iterated quantifiers. First, here are some rather trivial, but nevertheless valid inferences:

> $\forall x \forall y \varphi \leftrightarrow \forall y \forall x \varphi$ is valid.

Much more interesting, of course, are other combinations. For a start, it is easy to see that a quantifier combination $\forall x \exists y$ does not imply $\exists y \forall x$: everyone has a mother, but no one is a mother of everyone. And even $\forall x \exists y Rxy$ does not imply the same shape with variables permuted: $\forall x \exists y Ryx$. Everyone has a parent, but not everyone has a child. Here is about the only interesting valid inference between 2-quantifier combinations:

> $\exists x \forall y \varphi$ implies $\forall y \exists x \varphi$.

You may formulate for yourself why this is intuitively (and really) valid.

Exercise 4.17 Determine validity or invalidity of all the remaining possible implications between repeated quantifiers.

This ends our list of intuitively valid principles of reasoning with quantifiers.

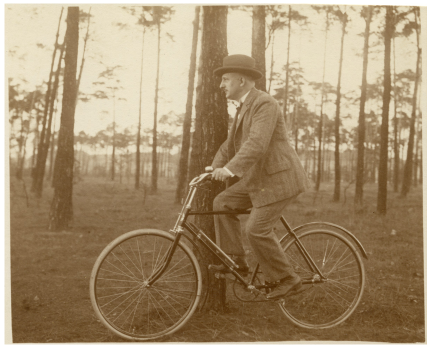
Warning: the role of compositionality We do not want to leave you with the wrong impression. From our progressive list, you might think that we have now classified 2-quantifier inferences, then we should go on to 3-quantifier ones, and so on. This was indeed how some medieval logicians saw the task ahead for logic. But this is mistaken. We do not have to go up the hierarchy that seemed in the making. There is a complete proof system for predicate logic whose rules just tell us explicitly what single quantifiers do. All the valid behaviour of complex quantifier combinations then follows automatically by finding the right *combinations* of proof steps for single quantifiers.

4.4 Formulas, Situations and Pictures

By now, it will have become clear to you that predicate logic is a very general language for talking about situations where objects have certain properties or are in certain relations. But what that really means is that these situations are crucial to understanding truth or falsity of formulas. Therefore, let us talk a bit more about what they are, intuitively, and how they connect up with formulas.

You can also see this point as follows. The semantics of a language ties the syntactic expressions of that language to real situations that language users talk about, say when communicating information to each other. Thus semantics has two independent pillars: language, and reality: the situations which that language can in principle describe.

Scenes from daily life Here is a simple example from everyday life:

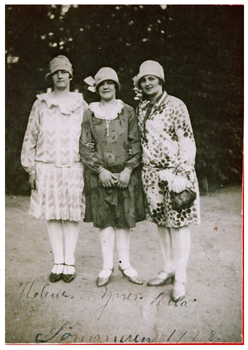


Suppose we use B for the property of being a bicycle, M for the property of being a man, and R for the relation of riding, then we can say that the following formula describes the following salient fact about the situation in the picture:

$$\exists x \exists y (Mx \wedge By \wedge Rxy).$$

Of course, you can also write statements about the trees.

Exercise 4.18 Consider the following picture:

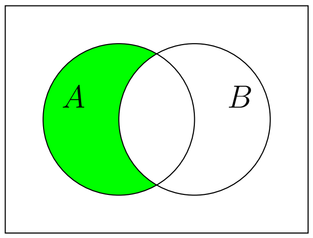


Use G for the property of being a girl, H for the property of being a hat, and W for the relation of wearing (so that Wxy expresses that x is wearing y). Now do the following:

(1) Give a predicate logical formula that is *true* for the situation in the picture.

(2) Give a predicate logical formula that is *false* for the situation in the picture.

Venn diagrams The Venn diagrams that you know from our chapter on syllogistic reasoning are abstract pictures of situations where objects have certain properties. Look at the Venn picture of the set of things that are in A but not in B:



(4.14)

Here is one relevant predicate logical formula:

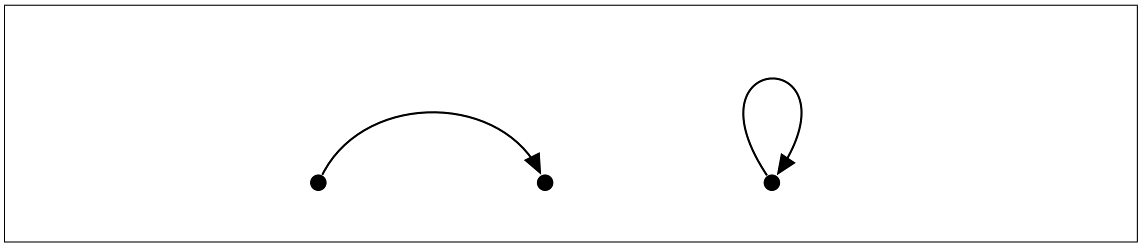
$$Ax \wedge \neg Bx. \tag{4.15}$$

The green area in the picture corresponds to the set of individuals x that make formula 4.15 true. Here, the formula (4.15) contains a *free variable* x, that is, x is *not bound* by a quantifier. This serves as a place-holder where different objects can be put to make true or false statements, and hence this formula describes a property of objects. Of course, this is exactly how mathematicians use variables as well: given the situation of the real numbers, the formula $x^2 < x$ defines a particular subset, namely the real numbers from 0 up to, but not including 1.

Graphs for representing relations A widely used concrete picture for a situation is a *graph*: a set of points connected by lines (often called an "undirected graph") or by directed arrows (a "directed graph"). Graphs are used everywhere in science, from pure mathematics to social networks, and also in daily life: the NS railway map of The Netherlands is a graph of cities and railway connections. In fact, they have already been used in this course itself! *Trees* are a special kind of graph with nodes and connections from nodes to their children, and you have already used trees to represent logical formulas.

Graphs are one of the most useful mathematical structures all around, being abstract yet simple to grasp. They are often used to demonstrate points about logic.

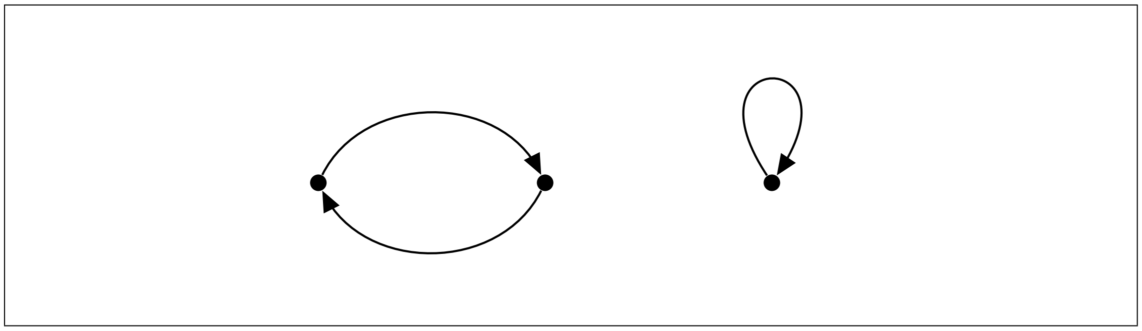
Here is a picture of a directed graph: a set of nodes, linked by arrows called "edges":



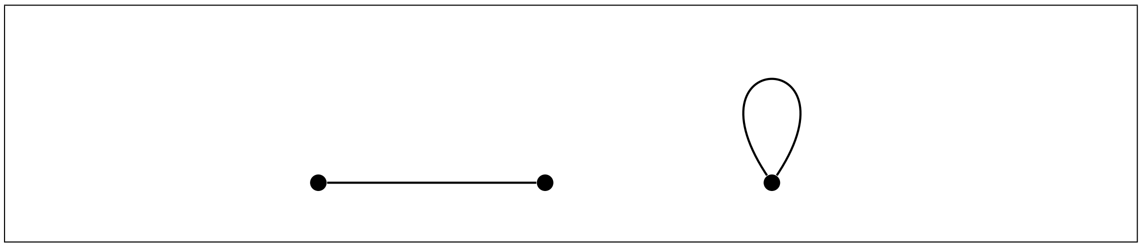
Let us take one binary predicate R for expressing that two objects in the picture are linked by an edge, with the arrow pointing from the first to the second object. Then it is easy to see that the following formulas are true:

$$\exists x \exists y Rxy$$
$$\exists x Rxx$$
$$\neg \forall x \exists y Rxy$$

Now let us change the picture by adding one more edge, to make the relation *symmetric*:



Effectively, this changes the picture into an *undirected* graph: since every arrow can be reversed, there is no need anymore to indicate the direction of the edges, and the picture can be simplified, as follows:



The formula $\forall x \forall y(Rxy \rightarrow Ryx)$, is not just true here, but is true in all undirected graphs: it expresses that the link relation is *symmetric*.

Symmetry is one of many important properties of binary relations that can be defined in predicate logic. We will see a few more in a moment.

Exercise 4.19 The formula

$$\forall x \forall z (\exists y (Rxy \land Ryz) \to Rxz)$$

expresses *transitivity* of the relation R. Which of the following relations are transitive:

(1) being an ancestor of ... on the set of human beings,

(2) being a parent of ... on the set of human beings,

(3) the 'less than' relation $<$ on the natural numbers,

Example 4.20 The binary relation in the last picture above is *not* transitive. Here is why not. Consider the left and the middle point in the picture. There is a link from left to middle, and there is a link from middle to left. This is so because, as we have just seen, the relation in the picture is symmetric. Now transitivity says that if you can get somewhere in two steps by following a relation, then you can also get there in a single step in that relation. Thus, transitivity demands that you can go from the left point to the left point in a single step. This is not the case, so the relation is not transitive.

Exercise 4.21 A relation R is called *linear* if the following holds:

$$\forall x \forall y (Rxy \lor x = y \lor Ryx).$$

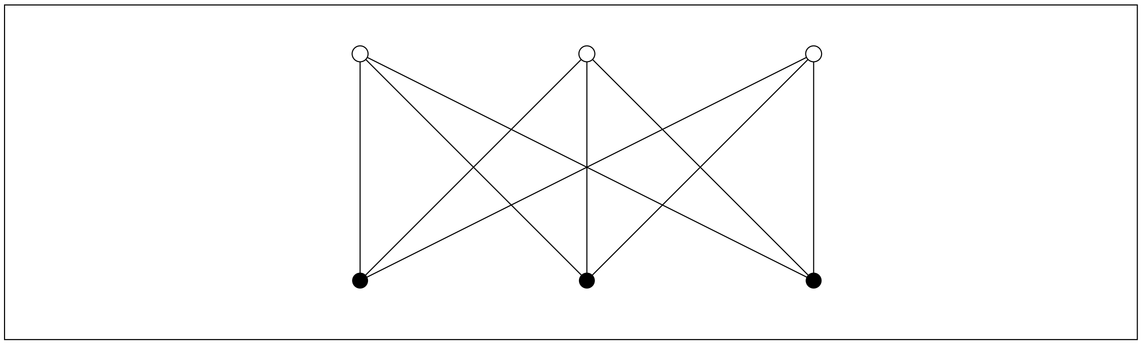
Clearly, the relation in the last picture above is not linear, for there is no link between the middle point and the point on the right in the picture. Which of the following relations are linear:

(1) being an ancestor of ... on the set of human beings,

(2) being a parent of ... on the set of human beings,

(3) the 'less than' relation $<$ on the natural numbers,

The next graph has a distinction between two kinds of nodes. This is something we can talk about with an additional 1-place predicate:



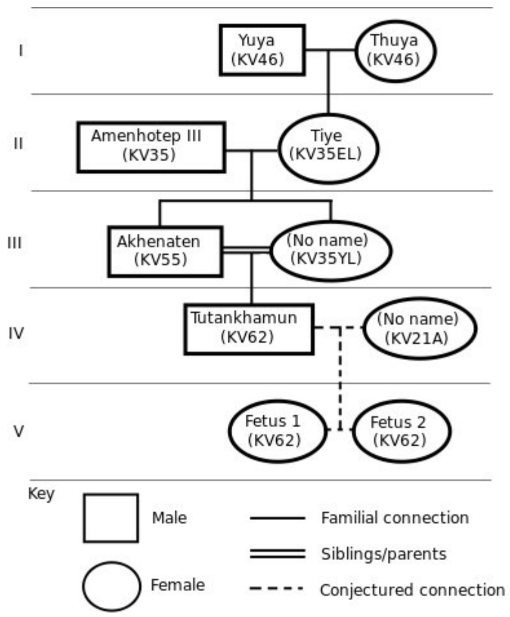
Let us say that the solid dots • have the property P, and the open dots ∘ lack that property. The true statement "All solid dots are linked to at least one open dot" can now be expressed as follows:

$$\forall x(Px \to \exists y(\neg Py \wedge Rxy)).$$

And here is how we express that links are only between solid and open dots:

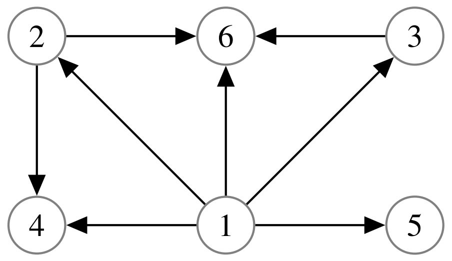
$$\forall x \forall y(Rxy \to (Px \leftrightarrow \neg Py)).$$

Most useful graphs in logic are directed, with arrows making links. A very nice example are family relations, as represented in genealogical *trees*. Here is the family tree of the famous Egyptian pharao Tutankhamun, as revealed by DNA research in 2010:



Exercise 4.22 Using relation symbols P for "parent of", and M for male, give a predicate logical formula that expresses the surprising fact (well, not so surprising, if you know something about ancient Egyptian royal families) that came to light about the lineage of Tutankhamun: *The parents of Tutankhamun were brother and sister.*

Directed graphs visualize relations of many kinds. A concrete example is *divisibility* on the natural numbers $\{1, 2, 3, 4, 5, 6\}$:



(4.16)

The circles around the numbers represent reflexive arrows: every number divides itself: $\forall x\,(x|x)$, where | is used as an infix symbol for the relation of divisibility. Another basic property of the divisibility relation is *transitivity*:

$$\forall x \forall y \forall z\,((x|y \wedge y|z) \rightarrow x|z).$$

This says that if one number divides a second, and that second a third, then the first number divides the third. Many relations are transitive: think of being older than, being as old as, being no older than (think about this!), being an ancestor of. Another general feature of divisibility is its *anti-symmetry*:

$$\forall x \forall y((x|y \wedge y|x) \rightarrow x = y)$$

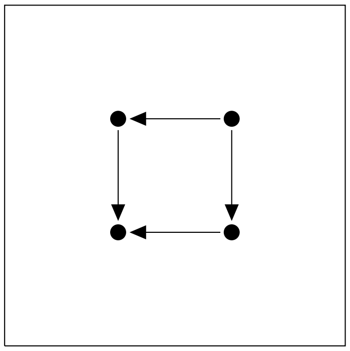
This, too, holds for many relations: though not for being "as old as": why not?

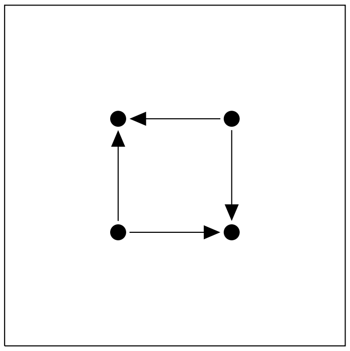
Example 4.23 We can use the predicate for "divides" to give a predicate logical definition of being a prime number, as follows. A prime number is a natural number with the property that it has no proper divisors greater than 1. A proper divisor of a number x is a number y that is smaller than x and that divides y. In predicate logic: y is a proper divisor of x greater than 1 if $1 < y \wedge y < x \wedge y|x$. Therefore, the following formula gives a definition of being prime:

$$P(x) \leftrightarrow \neg\exists y(1 < y \wedge y < x \wedge y|x).$$

You will encounter many more graphs later on in this course. Chapter 5 on knowledge and communication will have many undirected ones, Chapter 6 on action has directed graphs. We now state a few more exercises, since graphs concretely visualize much of what we are going to say more abstractly in the following sections.

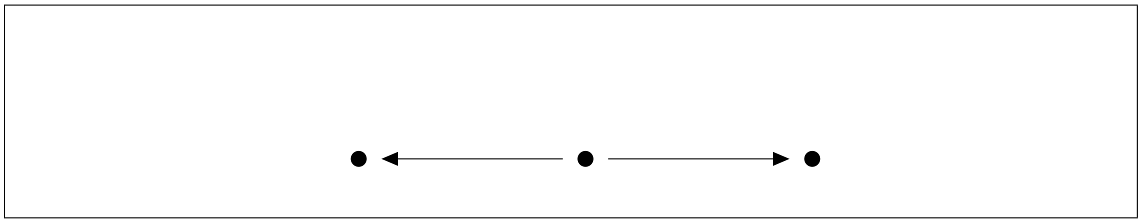
Exercise 4.24 Predicate logic can be used to tell two graphs apart: find a formula that is true in the one but not in the other. Consider the following two pictures:





Give a formula, with R for the relation, that is *true* on the left and *false* on the right.

Exercise 4.25 ♡ When a formula is false in a graph, we may want to *change* that graph in a minimal manner to make the formula true after all. Think of an engineer changing a blueprint to meet a given specification. Consider the following graph:



The formula $\exists x \forall y Rxy$ is false in the graph if R is interpreted as the $\rightarrow$ relation. Make the formula true by adding a single $\rightarrow$ link to the graph.

Exercise 4.26 ♠ Here is a harder problem. Suppose that we are talking about finite directed graphs whose ordering is reflexive and *connected*, where the latter property means that for any two points, an arrow runs from at least one to the other. (How would you write this as a formula?) Many communication networks are like this. Now call a point in the graph a "Great Communicator" if it can reach every point in the graph by one, or at most two successive arrows. (How would you write this in a formula?) Now the real question:

> Show that every finite connected graph has a "Great Communicator".

You may want to draw some pictures first to get a feeling for what these graphs are like. While we are at it, here is one more question. The stated inference does *not* hold for infinite graphs: can you give a counter-example?

We conclude with a few general observations on the link between predicate-logical formulas and situations (from numbers to pharao's) that we have now demonstrated in many concrete settings.

"The missing link": semantic interpretation Assigning meaning to a language really involves *three ingredients*, not just the two we discussed so far. We already had the sentences of the language, a syntactic code. Then we looked at the situations described, consisting of objects with structure: properties, relations, and the like. But the syntactic code only acquires meaning by *connecting* it with structures, a process of "interpretation". If you receive a letter in a language that you do not know, you just have the code. If you see a newspaper article in a language that you do not know, but with an inserted picture of Mount Etna, you may know the situation (yesterday's spectacular eruption) and the code, but you still do not know what the message says, since you do not know its interpretation. This three-way scheme gives language an amazing versatility. One piece of syntactic code can talk about many situations, and even about one situation under different interpretations, one situation can be described in many languages, and so on.

These ideas apply also to the language of predicate logic. As we have seen, it describes "universes of discourse"' consisting of objects, but we need a link. In the above, this link was often given informally as a "key" mapping predicate-logical symbols to corresponding structural components of the relevant situation. How does this linkage work in general?

In propositional logic, the link was the *valuation* mapping proposition letters to truth values. But this will no longer do. For checking whether a statement saying that a certain object has a certain property, or that certain objects are in a certain relation is true we need something more refined. Instead of just saying that "John is boy" is assigned the value *true*, we now need an interpretation for "John" and an interpretation for "being a boy". Here is a list, to give you a first impression of what we need:

(1) Individual constants denote concrete objects,much like proper names.

(2) Unary predicate letters denote concrete properties of objects, and likewise, binary predicate letters concrete binary relations, and so on. (We can think of these assigned predicates in terms of sets of objects or of ordered tuples and so on, but this is just a particular mathematical implementation, not the essence of the matter.)

(3) Variables will not denote fixed objects, but they can run through objects: something for which we will use a special device later on called an "assignment".

(4) Logical expressions like Boolean connectives and quantifiers are given their standard meaning that we have already seen at work throughout this text.

Interpreting complex sentences in stages Mapping the basic alphabet of the language to corresponding items in a situation allows us to interpret atomic sentences saying that one or more objects stand in some relation. What about more complex sentences formed with perhaps lots of logical operations, a set containing infinitely many expressions of extremely high syntactic complexity?

One of the many amazing things about predicate logic, and indeed, one of the major discoveries by its 19th century founding fathers, is that we do not have to worry about this. As we will see in more formal detail soon, formulas with nested quantifiers are constructed systematically by *stepwise syntax rules* introducing one symbol at a time. Next, these rules can be *interpreted semantically*, and then meanings for expressions of arbitrary complexity just arise automatically by calling on the meaning of single propositional operators and quantifiers as many times as required.

This process of stepwise interpretation from components is called *compositionality*, and it is a major design principle, not just in logic, but also, for instance, in programming languages or algebraic formalisms in mathematics. Compositionality is also important to philosophers, since it seems to provide an explanation for what is after all a very mysterious phenomenon. On the basis of only a finite amount of information (sample sentences from your parents, perhaps some grammatical rules in school) competent speakers of a language understand a potential infinity of new sentences when confronted with them, even when they have never seen them before.

4.5 Syntax of Predicate Logic

The time has come to confront the formal details of how predicate logic works. You have now seen informally how the language of predicate logic works in a number of settings, and how it can describe various structures from mathematics to our daily world. Next, you may want to see how it works for the same purposes that you have seen in earlier chapters, such as information update and, in particular, inference. But before we do that, it is time to sharpen up things, and say exactly what we mean by the language and semantics of predicate logic.

The first step is an investment in 'formal grammar'. It looks a bit technical, but still a far cry from the complexities of a real natural language like English. The following grammar is a simple model of basics of the grammar of many languages, including even programming languages in computer science.

Formulas As we have seen, predicate logic is an extension of propositional logic with *structured basic propositions* and *quantification*.

- An atomic formula consists of an n-ary predicate followed by n variables.
- A universally quantified formula consists of the symbol $\forall$ followed by a variable followed by a formula.
- An existentially quantified formula consists of the symbol $\exists$ followed by a variable followed by a formula.

- The rules for Boolean connectives are as in propositional logic.

Formal grammar We will now give a formal definition of the language of predicate logic. The definition assumes that individual terms are either variables or constants. We use a popular notation from computer science ("BNF format") for optimal perspicuity:

$$
\begin{array}{rcl}
\mathbf{v} & ::= & x \mid y \mid z \mid \cdots \\
\mathbf{c} & ::= & a \mid b \mid c \mid \cdots \\
\mathbf{t} & ::= & \mathbf{v} \mid \mathbf{c} \\
\mathbf{P} & ::= & P \mid Q \mid R \mid \cdots \\
\textbf{Atom} & ::= & \mathbf{P}\,\mathbf{t}_1 \cdots \mathbf{t}_n \text{ where } n \text{ is the arity of } \mathbf{P} \\
\varphi & ::= & \textbf{Atom} \mid \neg\,\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid (\varphi \leftrightarrow \varphi) \mid \\
 & & \forall\,\mathbf{v}\,\varphi \mid \exists\,\mathbf{v}\,\varphi.
\end{array}
$$

Here is how you read such a schema. The lines separating items stand for a disjunction of cases. Then, e.g., the clause for formulas says that a formula is either an atomic formula, or a negation of something that is already a formula, and so on. The class of formulas is understood to be the *smallest set of symbol sequences that is generated in this way*.

The following strings are formulas of the predicate logical language:

- $\neg Px$
- $(Px \wedge Qy)$
- $((Px \wedge Qy) \vee Rxz)$
- $\forall x Rxx$
- $\exists x(Rxy \wedge Sxyx)$

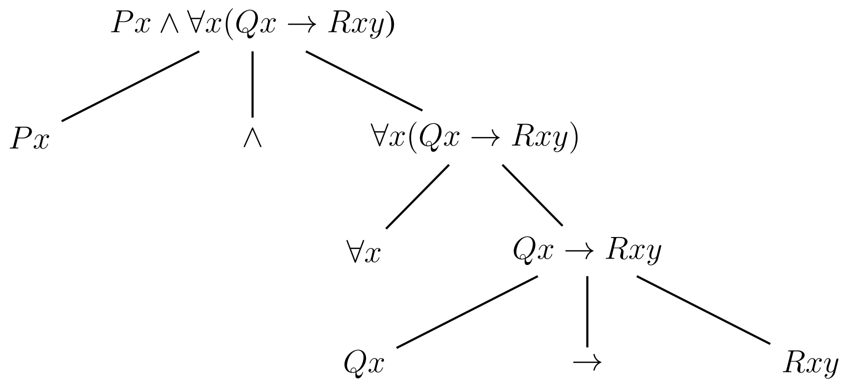
The semantics for this formal syntax will be given in the next section.

This official language definition places more parentheses than we have used so far. One usually omits parentheses when they are not essential for disambiguation.

We will now do a bit of "basic grammar" of predicate logic. The following may seem like a lot, but it is really very mild. Compare how much basic grammar you have to learn to use a natural language!

Free and bound variables Quantifiers and variables work closely together. In formulas we distinguish between the variable occurrences that are *bound* by a quantifier occurrence in that formula and the variable occurrences that are not. Binding is a syntactic notion, and it can simply be stated as follows. In a formula $\forall x\, \varphi$ (or $\exists x\, \varphi$), the quantifier occurrence binds all occurrences of x in φ that are not bound by any quantifier occurrence $\forall x$ or $\exists x$ inside φ.

As an example, consider the formula $Px \wedge \forall x(Qx \to Rxy)$. Here is its syntax tree:



The occurrence of x in Px is free, as it is not in the scope of a quantifier; the other occurrences of x (the one in Qx and in Rxy) are bound, as they are in the scope of $\forall x$. An occurrence of x is bound in φ if some quantifier occurrence binds it, and free otherwise.

Exercise 4.27 Give the bound occurrences of x in the following formula.

$$\exists x(Rxy \vee Sxyz) \wedge Px$$

Exercise 4.28 Which quantifier occurrence binds which variable occurrences?

$$\forall x(Px \to \exists x Rxx).$$

A predicate logical formula is called *open* if it contains at least one variable occurrence which is free (not bound by any quantifier); it is called *closed* otherwise. A closed predicate logical formula is also called a predicate logical *sentence*, which makes a complete assertion. $Px \wedge \exists x Rxx$ is an open formula, but $\exists x(Px \wedge \exists x Rxx)$ is a sentence.

Exercise 4.29 Which of the following formulas are sentences?

(1) $\forall x Px \vee \forall x Qx$,

(2) $Px \vee \forall x Qx$,

(3) $Px \vee Qx$,

(4) $\forall x(Px \to Rxy)$,

(5) $\forall x(Px \to \exists y Rxy)$.

If a formula φ has no free occurrences of x, quantifiers $\forall x\varphi$, $\exists x\varphi$ in it are called *vacuous*. While this may seem perverse in communication, technically, vacuous quantifiers do help in keeping the syntax and proof system of predicate logic smooth and simple.

Exercise 4.30 Which quantifications are vacuous? Replace each formula with vacuous quantification by an equivalent formula without vacuous quantification.

(1) $\forall x \exists x Rxx$.

(2) $\forall x \exists y Rxx$.

(3) $\forall x \exists y Rxy$.

(4) $\forall x \exists y Ryy$.

Substitution Now consider the ways we have for talking about objects. A *term* is either a constant or a variable. Thus, x is a term, and the constant c is also a term.

Here is a function that *substitutes* a term t for the occurrences of a variable v in a term s, with notation s^v_t:

$$\boxed{\begin{array}{rcl} c^v_t & := & c \\ v^v_t & := & t \\ {v_1}^v_t & := & v_1 \text{ for } v_1 \text{ different from } v \end{array}}$$

This follows the clauses constructing terms in the syntax of predicate logic.

Here is how this definition works out in a concrete case:

x^y_z is equal to x, x^x_c is equal to c, x^x_y is equal to y.

Next, we use the definition of s^v_t to define the widely used notion of *substitution of a term t for all free occurrences of a variable v in a formula φ*, with notation

$$\varphi^v_t.$$

This says that the property expressed by φ holds of the object denoted by t. This time, the definition follows the above syntactic clauses for constructing of the formulas of predicate logic:

$$\begin{array}{rcl}
(Pt_1 \cdots t_n)^v_t & := & P{t_1}^v_t \cdots {t_n}^v_t \\
(\neg\varphi)^v_t & := & (\neg\varphi^v_t) \\
(\varphi_1 \wedge \varphi_2)^v_t & := & ({\varphi_1}^v_t \wedge {\varphi_2}^v_t) \\
(\varphi_1 \vee \varphi_2)^v_t & := & ({\varphi_1}^v_t \vee {\varphi_2}^v_t) \\
(\varphi_1 \rightarrow \varphi_2)^v_t & := & ({\varphi_1}^v_t \rightarrow {\varphi_2}^v_t) \\
(\varphi_1 \leftrightarrow \varphi_2)^v_t & := & ({\varphi_1}^v_t \leftrightarrow {\varphi_2}^v_t) \\
(\forall v_1\varphi)^v_t & := & \begin{cases} \forall v_1\varphi & \text{if } v_1 \text{ equals } v, \\ \forall v_1\varphi^v_t & \text{otherwise.} \end{cases} \\
(\exists v_1\varphi)^v_t & := & \begin{cases} \exists v_1\varphi & \text{if } v_1 \text{ equals } v, \\ \exists v_1\varphi^v_t & \text{otherwise.} \end{cases}
\end{array}$$

Exercise 4.31 Give the results of the following substitutions:

(1) $(Rxx)^x_c$.

(2) $(Rxx)^x_y$.

(3) $(\forall xRxx)^x_y$.

(4) $(\forall yRxx)^x_y$.

(5) $(\exists yRxy)^x_z$.

Defining notions by recursion A remarkable feature of our grammatical definitions is their use of *recursion*. We explain what $(\neg\varphi)^v_t$ means by assuming that we already know what substitution does on smaller parts: φ^v_t, and so on. This recursion works because the definition follows precisely the construction pattern that was used for defining the formulas in the first place.

Alphabetic variants Using the notion of a substitution, we can say what it means that a formula is an *alphabetic variant* of another formula. This is useful since we often want to switch bound variables while retaining the essential structure of a formula.

Suppose φ does not have occurrences of z, and consider φ^x_z, the result of replacing all free occurrences of x in φ by z. Note that $\forall z\varphi^x_z$ quantifies over variable z in all places where $\forall x\varphi$ quantifies over x. We say that $\forall x\varphi$ and $\forall z\varphi^x_z$ are *alphabetic variants*.

Here are some examples: $\forall xRxx$ and $\forall yRyy$ are alphabetic variants, and so are $\forall x\exists yRxy$ and $\forall z\exists xRzx$. The quantification patterns are the same, although different variable bindings are employed to express them.

This section has given you some basic parts of the grammar of predicate logic. There are some other grammatical notions that are widely used, such as the 'quantifier depth' of a formula, being the longest 'nesting' of quantifiers that take scope over each other inside the formula. Again, this can be defined by recursion on the construction of the formulas, as given by the grammar definition: atomic formulas have quantifier depth 0, negations do not change depth, the depth of a conjunction is calculated as the maximum of the depths of its conjuncts, and similarly for the other binary boolean connectives, and finally a quantifier increases the depth by one. The following definition expresses this formally:

$$
\begin{aligned}
d(Pt_1 \cdots t_n) &:= 0 \\
d(\neg\varphi) &:= d(\varphi) \\
d(\varphi_1 \wedge \varphi_2) &:= \max(d(\varphi_1), d(\varphi_2)) \\
d(\varphi_1 \vee \varphi_2) &:= \max(d(\varphi_1), d(\varphi_2)) \\
d(\varphi_1 \rightarrow \varphi_2) &:= \max(d(\varphi_1), d(\varphi_2)) \\
d(\varphi_1 \leftrightarrow \varphi_2) &:= \max(d(\varphi_1), d(\varphi_2)) \\
d(\forall v\varphi) &:= 1 + d(\varphi) \\
d(\exists v\varphi) &:= 1 + d(\varphi)
\end{aligned}
$$

For the moment, you have seen enough grammatical precision, and we can proceed.

4.6 Semantics of Predicate Logic

Now it is time to say in more detail how the formal language of predicate logic gets interpreted systematically on semantic structures. We have said a lot already about these situations in Section 4.4, now the time has come to be more precise. The following notion packages together the notion of "structure" that we had before with the notion of an "interpretation" linking syntactic expressions in the language to matching parts of the structure.

Models We will give our definition for a special case, but once you grasp that, you will easily see how things work in general for the formal language that we have defined in the preceding section. Let us suppose that the predicate letter P has arity 1, R has arity 2, and S has arity 3. What should a relevant situation look like? A *model* $\mathcal{M}$ is a pair (D, I) with a *domain* D consisting of individual objects, together with an I that assigns the right kind of predicates on objects in D: a property for P, a binary relation for R, and a ternary relation for S. In set-theoretic notation, this may be written as follows:

$$\begin{array}{rcl} I(P) & \subseteq & D, \\ I(R) & \subseteq & D \times D, \\ I(S) & \subseteq & D \times D \times D. \end{array}$$

Here $D \times D$ (sometimes also written as D^2) is the set of all ordered pairs of objects in D, and $D \times D \times D$ (also written as D^3) is the set of all ordered triples of objects in D. Here is a useful notation. We write $I(P)$ as P_I, $I(R)$ as R_I, and $I(S)$ as S_I. This gives a clear distinction between a predicate letter P and the predicate P_I that interprets this predicate letter. We will not give new illustrations for such models: everything we have done in Section 4.4 is an illustration.

In general, for any predicate-logical language (with any sort of vocabulary), a model $\mathcal{M} = (D, I)$ has a non-empty domain D of objects plus an interpretation function I mapping the relation symbols of L to appropriate predicates on D, and the constants of L to objects in D.

Variable assignments Having dealt with predicates and constants, it remains to make sense of our final important piece of syntax: *variables* for objects. Intuitively, a model $\mathcal{M} = (D, I)$ suffices for giving truth values to sentences of our language, that make an assertion that is true or false in the model. But in doing so recursively, we cannot avoid dealing also with formulas containing free variables: unpeeling one quantifier will already leave us with a component formula of the latter sort.

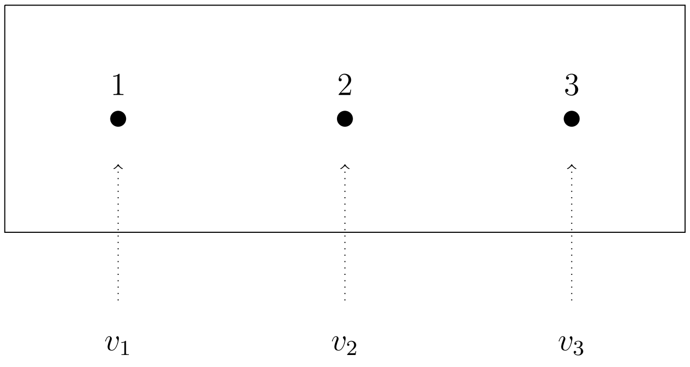
This calls for a new notion. Let V be the set of variables of the language:

A function g from variables to objects in D is called a *variable assignment*.

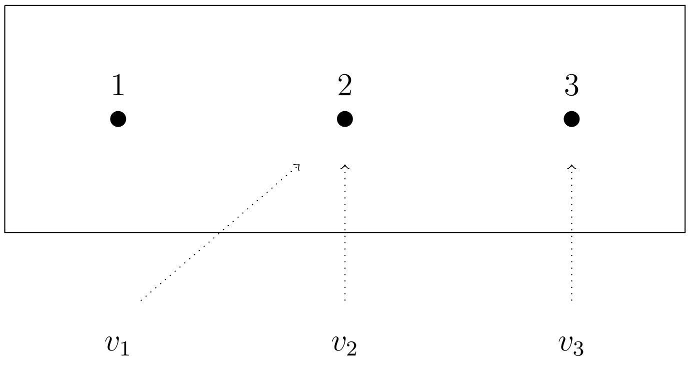
Later on, we will need to make minimal changes to variable assignments:

$g[v := d]$ is the variable assignment that is completely like g except that v now gets the value d (where g might have assigned a different value).

For example, let $D = \{1, 2, 3\}$, and let $V = \{v_1, v_2, v_3\}$. Let $g(v_1) = 1, g(v_2) = 2, g(v_3) = 3$. See the following picture, where the g links are indicated by dotted arrows.



Then $g[v_1 := 2]$ is the variable assignment that is like g except that v_1 gets the value 2, i.e. the assignment that assigns 2 to v_1, 2 to v_2, and 3 to v_3:



Assignments and computation We emphasize this picture of value change because it has found important other applications, after it was first proposed. In particular, the above pictures drive the semantics of imperative programming languages (a topic that will be explained in Chapter 6 below). An assignment models a *memory state* of a computer, where the variables are "addresses" that store "current values". A computer typically works by successively replacing values in addresses by new values – and that is precisely what the above assignment change does.

But right here, we put assignments to work in the semantics of predicate logic.

Truth definition for predicate logic At last, we are ready for the core of the semantics of predicate logic: the definition of truth of a formula in a model. This truth definition for predicate logic is due to the Polish logician Alfred Tarski (1901–1983):

Alfred Tarski

Let $\mathcal{M} = (D, I)$ be a model for predicate logical language L, and let g be a variable assignment for L in M. Consider any formula φ of L. We will assume for concreteness that L has a unary predicate symbol P, a binary R, and a ternary S, plus a constant symbol c. The general treatment will then be obvious from what follows. We are going to define the notion

$M \models_g \varphi$, for *φ is true in M under assignment g.*

We also sometimes say that *g satisfies φ in model M.*

Values of terms First, we consider the terms of the language: variables are interpreted by the variable assignment, constants using the "hard-wired" interpretation function of the model:

$$\begin{array}{rcl} [\![v]\!]_I^g & = & g(v) \\ [\![c]\!]_I^g & = & c_I \end{array}$$

Truth in a model Next, we define truth of a formula in a model given a variable assignment g. As usual, the definition follows the construction in the definition of predicate logical formulas in a stepwise manner:

$M \models_g Pt_1 \cdots t_n$	iff	$(\llbracket t_1 \rrbracket_I^g, \ldots, \llbracket t_n \rrbracket_I^g) \in P_I$
$M \models_g \neg\varphi$	iff	it is not the case that $M \models_g \varphi$.
$M \models_g \varphi_1 \wedge \varphi_2$	iff	$M \models_g \varphi_1$ and $M \models_g \varphi_2$
$M \models_g \varphi_1 \vee \varphi_2$	iff	$M \models_g \varphi_1$ or $M \models_g \varphi_2$
$M \models_g \varphi_1 \to \varphi_2$	iff	$M \models_g \varphi_1$ implies $M \models_g \varphi_2$
$M \models_g \varphi_1 \leftrightarrow \varphi_2$	iff	$M \models_g \varphi_1$ if and only if $M \models_g \varphi_2$
$M \models_g \forall v\varphi$	iff	for all $d \in D$ it holds that $M \models_{g[v:=d]} \varphi$
$M \models_g \exists v\varphi$	iff	for at least one $d \in D$ it holds that $M \models_{g[v:=d]} \varphi$

What we have presented just now is a recursive definition of truth for predicate logical formulas in given models with a given assignment. Note in particular how the clauses for the quantifiers involve the above-defined changing assignments.

Compositionality once more This formal definition implements the *compositionality* that we have discussed earlier. It shows how a small finite set of rules can interpret infinitely many formulas, of arbitrary syntactic complexity. In particular, if you think back of our earlier many-quantifier formulas, say on graphs, their meaning is completely described by the mechanism given here: *meaning of single words, repeated in tandem with formula structure*.

This completes our definition of the semantics: we now explore it a bit.

Finiteness property and closed formulas If we inspect how the truth definition works for complex formulas, the following *Finiteness Property* is easy to see:

> The truth value of a formula π in a model $\mathcal{M}$ under an assignment g only depends on the objects which g assigns to the *free variables* in φ: if two assignments agree on those free variables, they both make φ true, or both false.

In particular, then, if we evaluate closed formulas φ that have no free variables at all, they are either true under all assignments, or false under all of them. Hence we can just talk about their truth or falsity "simpliciter": writing $M \models \varphi$ iff there is *some* assignment g with $M \models_g \varphi$.

The importance of open formulas Still, even in evaluating a closed formula like

$$\forall x(Px \to \exists yRxy),$$

the above recursive clause for the universal quantifier $\forall x$ relies on truth for the formula $Px \to \exists y Rxy$, which is open, and hence assignments are crucial to dealing with that. But more than this: formulas with free variables are not a nuisance, they are highly important in their own right. We can think of them as defining *properties of* or relations between the objects that fill the free variable slots. This is why, in computer science, they are often used to *query* a given model, say, a data base with information about individuals. In such a setting, the question $\varphi(x)$? ("Which objects are φ-ing?") asks for a list of all objects in the model that have the property φ.

4.7 Valid Laws and Valid Consequence

Now we can look more formally at valid reasoning in predicate logic. Our earlier notions from Chapters 2, 3 return in this setting:

Valid laws A predicate logical formula φ is called *logically valid* if φ is true in every model under every assignment. Notation: $\models \varphi$.

Without further ado, here is how you can test your understanding of this:

Exercise 4.32 Which of the following statements are true?

(1) $\models \exists x Px \lor \neg \exists x Px$.

(2) $\models \exists x Px \lor \forall x \neg Px$.

(3) $\models \forall x Px \lor \forall x \neg Px$.

(4) $\models \forall x Px \lor \exists x \neg Px$

(5) $\models \exists x \exists y Rxy \to \exists x Rxx$.

(6) $\models \forall x \forall y Rxy \to \forall x Rxx$.

(7) $\models \exists x \exists y Rxy \to \exists x \exists y Ryx$

(8) $\models \forall x Rxx \lor \exists x \exists y \neg Rxy$.

(9) $\models \forall x Rxx \to \forall x \exists y Rxy$

(10) $\models \exists x Rxx \to \forall x \exists y Rxy$

(11) $\models \forall x \forall y \forall z((Rxy \land Ryz) \to Rxz)$.

Incidentally, our convention that the domains of our models are always non-empty is reflected in one particular law:

$$\models \forall x \varphi \to \exists x \varphi$$

Finally, as in propositional logic, we call two formulas φ, ψ *logically equivalent* if the formula $\varphi \leftrightarrow \psi$ is valid. Many practical uses of logic consist in reducing formulas to logical equivalents that are easier to grasp or manipulate.

Valid consequence More generally, we want to study the process of valid reasoning in predicate logic. When can we draw a conclusion ψ from premises $\varphi_1, \ldots, \varphi_n$? As in earlier chapters, our answer is this. Valid consequence says that the premises can never be true while the conclusion is false, or in other words, when the truth of the premises always brings the truth of the conclusion in its wake:

> A formula ψ *logically follows from* a formula φ (alternatively, φ *logically implies* ψ) if every model plus assignment which makes φ true also makes ψ true. The notation for 'φ logically implies ψ' is $\varphi \models \psi$.

The art of testing validity How can we see that statements of the form $\varphi \models \psi$ hold or do not hold? (where φ, ψ are closed formulas of predicate logic)? Sometimes a simple informal argument confirms the validity. And in principle, it is also clear how we can *refute* the statement $\varphi \models \psi$. We have to find a *counterexample*: that is a model $\mathcal{M}$ plus assignment g with $M \models_g \varphi$, but not $M \models_g \psi$ (the latter is often abbreviated as $M \not\models_g \psi$). Using your intuitive skills of this sort, try to answer the following questions:

Exercise 4.33 Which of the following statements hold? If a statement holds, then you should explain why. If it does not, then you should give a counterexample.

(1) $\forall x Px \models \exists x Px$

(2) $\exists x Px \models \forall x Px$.

(3) $\forall x Rxx \models \forall x \exists y Rxy$.

(4) $\forall x \forall y Rxy \models \forall x Rxx$.

(5) $\exists x \exists y Rxy \models \exists x Rxx$

(6) $\forall x \exists y Rxy \models \forall x Rxx$.

(7) $\exists y \forall x Rxy \models \forall x \exists y Rxy$

(8) $\forall x \exists y Rxy \models \exists y \forall x Rxy$.

(9) $\exists x \exists y Rxy \models \exists x \exists y Ryx$.

(10) $\forall x Rxx \models \forall x \exists y Rxy$.

(11) $\exists x Rxx \models \exists x \exists y Rxy$.

We can make all this slightly more general by allowing more than one premise. We say that a formula ψ logically follows from $\varphi_1, \dots, \varphi_n$ (in notation: $\varphi_1, \dots, \varphi_n \models \psi$) if for every model $\mathcal{M}$ for the language and assignment g with the property that $\mathcal{M} \models_g \varphi_1$ and ... and $\mathcal{M} \models_g \varphi_n$ it is also the case that $\mathcal{M} \models_g \psi$.

Exercise 4.34 Which of the following hold?

(1) $\forall x \forall y(Rxy \to Ryx), Rab \models Rba$

(2) $\forall x \forall y(Rxy \to Ryx), Rab \models Raa$

(3) $\forall x \forall y \forall z((Rxy \land Ryz) \to Rxz), Rab, Rac \models Rbc$,

(4) $\forall x \forall y \forall z((Rxy \land Ryz) \to Rxz), Rab, Rab \models Rac$,

(5) $\forall x \forall y \forall z((Rxy \land Ryz) \to Rxz), Rab, \neg Rac \models \neg Rbc$,

(6) $\forall x \forall y \forall z((Rxy \land Ryz) \to Rxz), \forall x \forall y(Rxy \to Ryx), Rab \models Raa$.

Maybe these exercises were not too difficult, but in general, testing for predicate-logical validity is much harder than the truth table method for propositional logic in Chapter 2, or the Venn diagrams for syllogistic reasoning in Chapter 3. We will devote all of Chapter 8 later in this course to a general method for testing validity for predicate logic called *semantic tableaux*. For now, we just explain what has become more difficult than before.

Mathematical complexity of the system Here are a few reasons why predicate-logical validity is much harder than that for propositional logic. One is that there are infinitely many models for the language. This infinity arises because domains of models can have many sizes: finite $0, 1, 2, 3, ...$ (and there are already infinitely many finite natural numbers), or even infinite (the real numbers are an infinite structure, and so are the usual geometrical spaces). This means that there is no finite enumeration of all relevant models to be checked, the way we did in truth tables or Venn diagrams.

In fact, it can even be *proved mathematically* that the notion of validity for predicate logic is harder than anything we have seen before:

> Validity for predicate logic is *undecidable*: there is no mechanical method at all that tests it automatically.

A proof of this result is beyond the scope of this course, but we will make some further comments on this topic in one of the Outlooks to this chapter.

This computational complexity may be seen as the price that we pay for the nice fact that predicate logic has much greater expressive power in its language than all the logical systems we have seen before. Still, some good news remains: we *can* axiomatize the validities of predicate logic in a formal *proof system* of the sort we have seen in earlier chapters, and our next section will say a little bit about how that works. But before we go there, here is one more relevant feature to understand.

Decidable parts of predicate logic The undecidability of predicate logic has to do with its full arsenal of expressive power: especially its use of binary and higher predicates, and its complex patterns of nested quantification. What that leaves open is the possibility that, inside the complex total system, there may be large decidable parts with lower expressive power that do have decision procedures for validity, and that might still be useful. And indeed there are many such "decidable fragments" of predicate logic, with new ones being discovered all the time.

One classical example of a decidable part of predicate logic is *monadic predicate logic* (see page 4-13 above). This is the system that arises from the above presentation when we keep the syntax and semantics as we gave them, but make one sweeping restriction: all predicate letters occurring in atomic statements must be *unary*, with just one argument place. What can we say in such a fragment? Well, it is easy to see that our earlier predicate-logical translation of syllogistic forms ended up inside this fragment. But so do many other inferences about properties and kinds. Now here is a result that we state without proof:

> Monadic predicate logic has a decision method for validity.

This is actually not so hard to see, since the structure of monadic formulas is easy to analyze. In particular, one can prove that in this fragment, each formula is logical equivalent to one without nested quantifiers. As a result we can even restrict ourselves to using finite models up to a bounded size in testing validity for the monadic fragment – something you may view as a sort of extended Venn diagram method. With these perhaps too mysterious hints, we leave this topic here.

4.8 Proof

The valid laws of predicate logic can be described as the provable theorems of a formal calculus like the ones you have seen in earlier chapters. Here is its list of principles:

(1) All propositional tautologies.

(2) $\forall x\varphi \rightarrow \varphi^x_t$. Condition: no variable in t occurs bound in φ.

(3) $\forall x(\varphi \rightarrow \psi) \rightarrow (\forall x\varphi \rightarrow \forall x\psi)$.

(4) $\varphi \rightarrow \forall x\varphi$. Condition: x does not occur free in φ.

Next, the system includes a *definition* of the existential quantifier as follows:

$$\exists x\varphi \leftrightarrow \neg\forall x\neg\varphi.$$

Finally, as our deductive proof rules, we have Modus Ponens as in propositional logic, plus a rule UG of *Universal Generalization*:

- MP: from φ and $\varphi \to \psi$, infer ψ.
- UG: from φ infer $\forall x \varphi$. Condition: x does not occur free in any premise which has been used in the proof of φ.

A *theorem* is now any formula that can be derived from the given axioms and definition by the stated rules in a finite number of steps. Here are two examples:

From our Axiom 2 of "universal instantiation" we can derive a dual axiom of "existential generalization":

1.	$\forall x \neg\varphi \to \neg\varphi^x_t$	axiom 2
2.	$(\forall x \neg\varphi \to \neg\varphi^x_t) \to (\varphi^x_t \to \neg\forall x \neg\varphi)$	propositional tautology
3.	$\varphi^x_t \to \neg\forall x \neg\varphi$	from 1,2 by MP
4.	$\varphi^x_t \to \exists x \varphi$	from 3, by def of $\exists$

Our second example of using this proof system employs hypothetical reasoning to prove an implication: first derive a conclusion C from a hypothesis H, and then conclude that $H \to C$ is a theorem. We start with two hypotheses, $\varphi \to \forall x \psi$ and φ, and we assume that t x is not free in φ. Next, we drop the hypotheses one by one.

1.	$\varphi \to \forall x \psi$	hypothesis
2.	φ	hypothesis
3.	$\forall x \psi$	MP from 1,2
4.	$\forall x \psi \to \psi$	axiom 2, with x for t
5.	ψ	MP from 3,4
6.	$\varphi \to \psi$	drop hypothesis 2
7.	$\forall x(\varphi \to \psi)$	UG of 6
8.	$(\varphi \to \forall x \psi) \to \forall x(\varphi \to \psi)$	drop hypothesis 1

Note that the use of UG in step 7 is justified by our assumption that x is not free in φ, and therefore, x is not free in the hypothesis $\varphi \to \forall x \psi$. The derivation shows that

$$\vdash (\varphi \to \forall x \psi) \to \forall x(\varphi \to \psi),$$

on condition that x is not free in φ.

As earlier in this course, we are not going to train you in finding formal proofs, but you may find it rewarding to learn how to read them – and perhaps even look for some simple ones by yourself. For now, we conclude with some background.

System properties This proof system has two features that you have seen earlier. First, we have

> *Soundness*: Every theorem of predicate logic is valid.

This is easy to see: the stated axioms are all valid, and you can check in the above list. Moreover, the proof rules are such that they derive valid formulas from valid formulas. Much more complex, and in fact about the first really deep result in modern logic is

> *Completeness*: Every valid formula is a provable theorem.

A proof of this important result (due to Kurt Gödel in his 1930 dissertation) would typically be your passport to a next-level more advanced logic course.

Kurt Gödel

But let us end this theoretical passage with a connection with the preceding section. Predicate logic is undecidable, but it does have a complete proof system. Is not there a tension between these two results? Cannot we use the above simple proof system, at least in principle, to test for validity? Say, we write down a formula, and we *enumerate all possible proofs* in the above deductive system in a long sequence, say in ascending length. (By the way, note that this enumeration sequence will be *infinite*: there are infinitely many theorems produced by the above calculus. Why?) Now if a formula is valid, we *must* encounter it somewhere at some finite stage along this sequence, and we are done... Here is the flaw in this reasoning. A method for testing should tell us after a finite number of steps whether or not the formula is valid. But in our proof system, the only way we can see that a formula φ is *not valid* is by running through the entire infinite sequence, and noting that it nowhere lists a proof for φ. This takes infinitely many time steps, and hence it is not a decision method.

Exercise 4.35 Here is a result known as "Post's theorem": Suppose that you have an infinite enumeration of all valid formulas of a logic, and also an infinite enumeration of all non-valid formulas. Then the system has a decision method. Show how that method would work.

4.9 Identity, Function Symbols, Algebraic Reasoning

Identity First we add identity (or: 'equality') to predicate logic. We start with a simple example. Define a 'celebrity' as a person who knows no one, but is known by everyone. This means, of course, someone who does not know anyone else, but is known by everyone else. To express this we need a predicate for identity (or: equality). So let's use the predicate $x = y$ that was mentioned before. Now we can say (employing the useful abbreviation $x \neq y$ for $\neg x = y$):

x is a celebrity (i) $\forall y(x \neq y \rightarrow (Kyx \wedge \neg Kxy))$.
(ii) $\forall y(x \neq y \rightarrow Kyx) \wedge \forall y(x \neq y \rightarrow \neg Kxy)$.

The two translations (i) and (ii) express the same thing: they are logically equivalent. The following formula expresses that C is a definition of the property of being a celebrity:

$$\forall x(Cx \leftrightarrow \forall y(x \neq y \rightarrow (Kyx \wedge \neg Kxy))). \tag{4.17}$$

From definition (4.17) and the formula that expresses that Mary knows John, Kmj, it follows that $\neg Cm$ (Mary is not a celebrity). Identiy is also needed to translate the following example:

John loves Mary and Bill loves another girl. (4.18)

Clearly, with "another girl" is meant "a girl different from Mary." Using identity we can translate the example as follows:

$$Ljm \wedge \exists x(Gx \wedge x \neq m \wedge Lbx). \tag{4.19}$$

Identity is a binary predicate with a fixed logical meaning. It is always written in infix notation, as $x = y$, and with $\neg x = y$ abbreviated as $x \neq y$. Here is a similar case where identity is useful:

Mary is the only girl that John loves. (4.20)

Being the only girl with a certain property means that all other girls lack that property. Spelling this out we get:

$$Gm \wedge Ljm \wedge \forall x\,((x \neq m \wedge Gx) \rightarrow \neg Ljx) \tag{4.21}$$

By using an equivalence instead of an implication symbol we can express this a bit shorter:

$$\forall x\,(Gx \rightarrow (x = m \leftrightarrow Ljx)) \wedge Gm \tag{4.22}$$

Rephrasing this in English it says that all girls have the property that being loved by John boils down to the same thing as being equal to Mary.

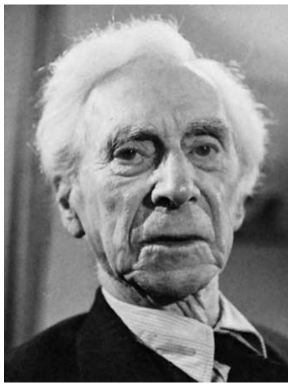
Identity provides a way of expressing *uniqueness properties* of objects. The following formula states that there is *exactly one* object which has the property P:

$$\exists x\,(Px \wedge \forall y\,(Py \rightarrow y = x)) \tag{4.23}$$

In words, there exists a 'P' such that each second object which has the property P as well must be equal to the first. Again, this can be formulated somewhat shorter by using an equivalence:

$$\exists x \forall y\,(Py \leftrightarrow y = x) \tag{4.24}$$

There is an object x such that for each object having the property P is the same thing as being equal to x.


Bertrand Russell

The British philosopher Bertrand Russell proposed to use this recipe to translate definite descriptions.

The father of Charles II was executed. (4.25)

This can be translated, using F for the property of being father of Charles II, as:

$$\exists x(Fx \land \forall y(Fy \to x = y) \land Ex), \tag{4.26}$$

or, as we have seen, more succinctly as:

$$\exists x(\forall y(Fy \leftrightarrow x = y) \land Ex). \tag{4.27}$$

Russell's analysis then gets extended to an account of the meaning of proper names as disguised or abbreviated descriptions.

In a similar way we can give a formula stating that there exists exactly two objects which have the property P:

$$\exists x \exists y\,(\neg x = y \land \forall z\,(Pz \leftrightarrow (z = y \lor z = x))) \tag{4.28}$$

This says that there exists a pair (of two *different* things) such that having the property P is the same as being equal to one of the members of this pair. To avoid such long re-formulation of relatively simple information (4.28) is sometimes abbreviated as $\exists!^2 x\, Px$, and in general $\exists!^n Px$ is used to express that exactly n things have the property P. In the case $n = 1$ we also write $\exists! x\, P$.

Exercise 4.36 Write out the predicate logical translation for “there are exactly three objects with property P”.

Exercise 4.37 The $\exists!^2$-quantifier can be used to give a predicate logical description of prime numbers in the divisibility graph as depicted in (4.16) on page 4-22. How? And what kind of numbers do you get when you replace $\exists!^2$ in your definition by $\exists!^3$?

Exercise 4.38

(1) Explain why the two following sentences have different meanings (by a description of a situation in which the sentences have a different truth-value)

> There is exactly one boy who loves exactly one girl. ($\exists! x\, \exists! y\, (Bx \wedge Gy \wedge Lxy)$)
>
> There is exactly one girl who is loved by exactly one boy. ($\exists! y\, \exists! x\, (Bx \wedge Gy \wedge Lxy)$)

(2) Explain why $\exists! x\, (Ax \vee Bx)$ and $\exists! x\, Ax \vee \exists! x\, Bx$ have different meanings.

Function symbols There is still one other device that predicate logic has for speaking about objects: function symbols. Mathematics is replete with *functions* that create new objects out of old, and likewise, predicate logic becomes more expressive if we allow function symbols for functions like ‘the square of x’ or ‘the sum of x and y’. We will see later how this can be used to write algebraic facts and do calculations, but for starters, note that function symbols also make sense for natural language. On the domain of human beings, for instance, the *father* function gives the father of a person. Suppose we write $f(x)$ for the father of x and $m(x)$ for the mother of x. Then we can say things like:

Paula’s father loves her	$Lf(p)p$
Bill and John are full brothers	$f(b) = f(j) \wedge m(b) = m(j)$
Not everyone has the same father	$\neg \forall x \forall y f(x) = f(y)$

In the lingo of mathematics, function symbols are combined with equality to express algebraic equations like the following:

$$x \cdot (y + z) = x \cdot y + x \cdot z. \tag{4.29}$$

This is a statement of the sort that you learn to manipulate in high school. In fact, this can be viewed as a formula of predicate logic. There is the equality predicate here, but the main point is made with the function symbols for addition and multiplication.

Finally, let us see what changes to the truth definition are needed to take the identity relation and function symbols into account. Identity statements are treated by adding the following clause to the semantics definition:

$$M \models_g t_1 = t_2 \quad \text{iff} \quad [\![t_1]\!]_I^g = [\![t_2]\!]_I^g$$

This treats identity as a special binary predicate, with a fixed interpretation: $t_1 = t_2$ is true if and only if t_1 and t_2 get interpreted as the same object in the model under consideration.

To extend the semantics to function symbols, we have to extend the truth definition to structured terms (terms containing function symbols). For that, we have to assume that the interpretation function I knows how to deal with a function symbol.

If f is a one-place function symbol, then I should map f to a unary operation on the domain D, i.e. to a function $f_I \colon D \to D$. In general it holds that if f is an n-place function symbol, then I should map f to an n-ary operation on the domain D, i.e. to a function $f_I \colon D^n \to D$. We use C for the set of zero-place function symbols; these are the constants of the language, and we have that if $c \in C$ then $c_I \in D$.

The interpretation of terms, given an assignment function g for variables and an interpretation function I for constants and function symbols, is now defined as follows. Note the by now familiar use of recursion in the definition.

$$[\![t]\!]_I^g = \begin{cases} t_I & \text{if} \quad t \in C, \\ g(t) & \text{if} \quad t \in V, \\ f_I([\![t_1]\!]_I^g, \ldots, [\![t_n]\!]_I^g) & \text{if} \quad t \text{ has the form } f(t_1, \ldots, t_n). \end{cases}$$

Algebraic reasoning In a domain of numbers (natural numbers, integers, fractions, reals) $+$ is a function symbol that takes two numbers and creates a number. The symbol $\cdot$ (also written as $\times$) is also a binary function symbol for the same domain of discourse.

Algebraic equations typically say that two complex terms, viewed as two different instructions for computation, denote the same object. If x, y, z refer to numbers, then $x \cdot (y + z)$ also refers to a number, and equation 4.29 given above states that $x \cdot y + x \cdot z$ refers to the same number.

Figure 4.1 gives a general set of equations that describes the interaction of addition and multiplication in a so-called *ring*, for a language with constants 0 and 1, and with binary functions $+$ and $\cdot$ (called binary operators) and unary function $-$. It is left to you to check that the equations hold, if we interpret the variables as integer numbers, but also if we interpret them as floating point numbers (fractional numbers, rational numbers), and also if we interpret the variables as real numbers (but do not worry if the difference between rational and real numbers escapes you).

$$\begin{aligned}
x + 0 &= x \\
x + (-x) &= 0 \\
x + y &= y + x \\
x + (y + z) &= (x + y) + z \\
x \cdot 1 &= x \\
1 \cdot x &= x \\
x \cdot (y \cdot z) &= (x \cdot y) \cdot z \\
x \cdot (y + z) &= x \cdot y + x \cdot z \\
(x + y) \cdot z &= x \cdot z + y \cdot z
\end{aligned}$$

Figure 4.1: Axioms for the interplay of $+$ and $\cdot$ in rings.

$$\begin{aligned}
x \vee y &= y \vee x \\
x \wedge y &= y \wedge x \\
x \vee (y \vee z) &= (x \vee y) \vee z \\
x \wedge (y \wedge z) &= (x \wedge y) \wedge z \\
x \wedge (x \vee y) &= x \\
x \vee (x \wedge y) &= x \\
x \wedge (y \vee z) &= (x \wedge y) \vee (x \wedge z) \\
x \vee x' &= 1 \\
x \wedge x' &= 0 \\
x \vee 1 &= 1 \\
x \wedge 0 &= 0
\end{aligned}$$

Figure 4.2: Laws of Boolean Algebra.

The same syntax with function symbols works everywhere in mathematics. Think of mathematical notation for sets. The domain of discourse now is a universe of sets. Letters x, y, ..., stand for arbitrary sets, $\emptyset$ is the special constant name of the empty set, and function terms for intersection and union create complex terms like $x \cap (y \cup z)$ that denote sets in the way you have already seen when computing pictures for Venn Diagrams and related structures.

We can also look at propositional logic in an algebraic way. The equations in Figure 4.2 describe the laws of Boolean algebra, with binary operators $\wedge, \vee$, a unary operator $'$ for complement (negation), and constants 1 and 0. It is left to you to check that these equations are indeed valid laws of propositional logic, if we read identity as propositional equivalence, 1 as truth, and 0 as falsity.

4.10 Outlook — Mathematical Background

Using the proof system we presented in Section 4.8, you can talk about all models of predicate logic, for a given choice of predicate letters and function symbols. But it is also common to *add* axioms, in order to talk about specific topics. As you may already have guessed, predicate logic can be used to talk about *anything*.

Mathematical Theories: Arithmetic Suppose we want to talk about addition on the natural numbers. This is the aritmethic of first grade, the stuff that comes before the tables of multiplication. The domain of discourse is

$$\mathbb{N} = \{0, 1, 2, 3, 4, \ldots\}.$$

The predicate logical language that we can use for this has a constant for zero (we will use $\mathbf{0}$ for this; the intention is to use $\mathbf{0}$ as a name for the number 0), and two function symbols, one for taking the successor of a number (we will use s for this), and the familiar $+$ for addition. The successor of the number is the number immediately following it.

We have four axioms and an axiom scheme (a recipe for constructing further axioms).

The first axiom states that 0 is not the successor of any number:

$$\forall x \neg sx = \mathbf{0}. \tag{PA1}$$

The second axiom states that different numbers cannot have the same successor, or stated otherwise, that if $sx = sy$ then $x = y$.

$$\forall x \forall y (sx = sy \to x = y). \tag{PA2}$$

Note that $sx = sy \to x = y$ is equivalent to $x \neq y \to sx \neq sy$, so this does indeed express that different numbers have different successors.

The third axiom expresses that adding zero to a number doesn't change anything:

$$\forall x\, x + \mathbf{0} = x. \tag{PA3}$$

The fourth axiom expresses a sum of the form $x + sy$ as the successor of $x + y$:

$$\forall x \forall y\, x + sy = s(x + y). \tag{PA4}$$

The third and fourth axiom together define addition for any pair of numbers x and z, as follows. Either z is equal to 0, and then apply (PA3) to see that the outcome is x, or z is the successor of another number y and then apply (PA4) to see that the outcome is the successor of $x + y$. This is called a *recursive* definition, with recursion on the structure of y.

Below we will see how the recursion definition of addition will help us in constructing inductive proofs of facts about the addition operation.

The final axiom takes the form of a scheme. Assume that $\varphi(x)$ is a formula with x free. Then the following is an axiom:

$$(\varphi(\mathbf{0}) \wedge \forall x(\varphi(x) \to \varphi(sx))) \to \forall y \varphi(y). \tag{PA5–scheme}$$

This axiom scheme expresses mathematical induction on the natural numbers. If you are unfamiliar with mathematical induction you might wish to consult section A.6 in the Appendix.

The theory of arithmetic defined by (PA1) — (PA5-scheme) is known as *Presburger arithmetic*, after Mojzesz Presburger, student of Alfred Tarki.

Mojzesz Presburger

In the language of Presburger arithmetic one cannot express properties like being a prime number, or define the relation of divisibility. But one can express properties like being even, being odd, being a threefold, and so on. As an example, the following formula of Presburger arithmetic expresses the property of being even:

$$\exists y\, y + y = x. \qquad (x \text{ is even})$$

And the property of being odd:

$$\exists y\, s(y + y) = x. \qquad (x \text{ is odd})$$

Exercise 4.39 Express the property of x of being a threefold in the language of Presburger arithmetic.

Adding multiplication Presburger arithmetic is the restriction of the arithmetical theory of addition and multiplication to just addition. If we extend the language with an operator $\cdot$ for multiplication, the properties of multiplication can be captured by a recursive definition, as follows:

$$\forall x\, x \cdot \mathbf{0} = \mathbf{0}. \qquad \text{(MA1)}$$

$$\forall x \forall y\, x \cdot sy = (x \cdot y) + x. \qquad \text{(MA2)}$$

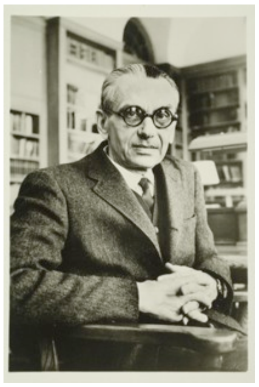
This defines $x \cdot y$ by recursion on the structure of y: MA1 gives the base case, MA2 the recursion case. In still other words: MA1 gives the first row in the multiplication table for x, and MA2 gives the recipe for computing the next row from wherever you are in the table, as follows:

$$\begin{array}{l} 0 \\ x \\ x + x \\ x + x + x \\ \vdots \end{array}$$

As you may have realized in second grade, the multiplication tables are constructed by means of addition.

Next, let IND be the induction axiom scheme for the language extended with multiplication. Thus, IND looks like PA5-scheme above, but for the extended language.

Giuseppe Peano


Kurt Gödel

The theory consisting of PA1 – PA4 plus MA1, MA2 and IND is called *Peano arithmetic*, after Giuseppe Peano (1858–1932). Here the logical situation is dramatically different from the case of Presburger arithmetic. One of the most famous results in logic is Kurt Gödel's incompleteness proof (1931) for this predicate logical version of arithmetic:

> Any predicate logical theory T that is an extension of Peano arithmetic is incomplete: it is possible to find formulas in T that make true statements about addition and multiplication on the natural numbers, but that cannot be proved in T.

Alonzo Church

Alan Turing

Five years later Alonzo Church and Alan Turing proved (independently) that the predicate logical language of arithmetic is undecidable, and more generally, that any predicate logical language with at least one 2-place relation symbol is undecidable (see Chapter 10).

4.11 Outlook — Computational Connection

Domains of discourse in programming The domains of discourse of predicate logic also play an important role in programming. In many programming languages, when a programming variable is declared, the programmer has to specify what kind of thing the variable is going to be used for.

In programming terms: variable declarations have to include a *type declaration*. A Java declaration `int i j max` declares three variables of type *int*, and an *int* in Java is a signed integer that can be stored in 32 bits: Java *int* variables range from $-2^{31} = -2,147,483,648$ to $2^{31} - 1 = 2,147,483,647$. What this means is that the type declarations fix the domains of discourse for the programming variables.

In programming, *types* are important because once we know the type of a variable we know how much storage space we have to reserve for it. More importantly, type information can be used to find common programming errors resulting from instructions to perform operations that violate type constraints, such as an attempt to add a character to an integer. Typing information for strongly typed languages such as Haskell is expressed in (a fragment of) predicate logic.

Statement of pre- and postconditions Predicate logic can also be used to state pre- and postconditions of computational procedures. Consider the following function (in Ruby) for computing an integer output from an integer input, together with a precondition and a postcondition:

```
# precondition: n >= 0
def ld(n)
 d = 2
 while d**2 <= n
   return d if n.remainder(d) == 0
   d = d + 1
 end
 return n
end
# postcondition:
# ld(n) is the least positive integer that divides n
```

The precondition is a predicate logical formula without quantifiers:

$$n \geq 0.$$

The postcondition can be translated into a predicate logical formula, using | for divides, as follows (we assume that the domain of discourse is $\mathbb{Z}$, the domain of integers):

$$\text{ld}(n)|n \land \forall m(0 < m < \text{ld}(n) \rightarrow \neg m|n).$$

The intended meaning is: if the input of the function satisfies the precondition then the output of the function will satisfy the postcondition.

Pre- and postconditions can be used for proving computational procedures correct. Examples of such correctness reasoning will be discussed in section 6.10.

Explicit statement of pre- and postconditions is also the key ingredient of a programming style called *design by contract*, the idea being that the precondition makes explicit what a computational procedure may assume about its input, while the postcondition is a statement about what the procedure is supposed to achieve, given the truth of the precondition. Preconditions state rights, postconditions list duties.

Exercise 4.40 (You should only attempt this if you have some programming experience.) Suppose the procedure for `ld` would have been stated like this:

```
# precondition: n >= 0
def ld(n)
 d = 2
 while d**2 < n
   return d if n.remainder(d) == 0
   d = d + 1
 end
 return n
end
# postcondition:
# ld(n) is the least positive integer that divides n
```

Why would this version of the program not satisfy the contract anymore? Try to find a simple counterexample.

4.12 Outlook — Predicate Logic and Philosophy

Historically, the development of predicate logic has had considerable influence on philosophy. The founding father of modern predicate logic, Gottlob Frege, was deeply convinced that ordinary language is inadequate for rigorous scientific thinking. Here is his famous comparison between ordinary language and his 'ideography' (what he means is: the formal language he proposes, which is essentially the language of predicate logic):

> I believe that I can best make the relation of my ideography to ordinary language clear if I compare it to that which the microscope has to the eye. Because of the range of its possible uses and the versatility with which it can adapt to the most diverse circumstances, the eye is far superior to the microscope. Considered as an optical instrument, to be sure, it exhibits many imperfections, which ordinarily remain unnoticed only on account of its intimitate connection with our mental life. But, as soon as scientific goals demand great sharpness of resolution, the eye proves to be insufficient. The microscope, on the other hand, is perfectly suited to precisely such goals, but that is just why it is useless for all others. [Fre76]

Frege believes that his proposal of a language of thought is important for philosophy precisely because it exhibits the limitations of ordinary language:

> If it is one of the tasks of philosophy to break the domination of the word over the human spirit by laying bare the misconceptions that through the use of language often almost unavoidably arise concerning the relations between concepts and by freeing thought from that with which only the means of expression of ordinary language, constituted as they are, saddle it, then my

> ideography, further developed for these purposes, can become a useful tool for the philosopher. [Fre76]

This point of view has been enormously influential. Philosophers like Bertrand Russell and Willard Quine concluded from the advances in logic that they had witnessed in their days that an important task for philosophy is to point out and repair fallacies that are due to the sway of natural language over the mind.

To focus on a light-hearted example, consider the following famous exchange:

> 'I see nobody on the road,' said Alice.
> 'I only wish I had such eyes,' the King remarked in a fretful tone. 'To be able to see Nobody! And at that distance too!'
> *Alice in Wonderland* [Car65]

The natural language syntax for "Nobody is on the road" sees this as a subject "Nobody" combined with a predicate "is on the road". But this natural language syntax does not correspond to the logical form, for the predicate logical translation of the sentence has no constituent corresponding to the subject:

$$\neg\exists x(\text{Person}(x) \wedge \text{OnTheRoad}(x)).$$

Though Frege's views fit in a long tradition of distinguishing between linguistic 'surface form' and its underlying 'logical form', Russell, Quine and other famous philosophers saw it as a battle call. They took this to mean that classical philosophy needed a total overhaul, since many of its assertions (say metaphysical ones about what sort of abstract objects exist in the world) were deeply infected with uncritically adopted natural language. Whether this accusation was true, is of course another matter. In fact, right until the present, there remains a difference in style between 'formal language philosophers' and 'natural language philosophers'.

Incidentally, in modern approaches to natural language analysis this problem is resolved by giving the subject a more articulate treatment: "nobody" refers to the set of all properties that no person has:

$$\{Y \mid \neg\exists x(\text{Person}(x) \wedge Y(x)\}. \qquad (4.30)$$

Since "being on the road" obviously refers to a property, we can view the syntactic operation of combining a subject with a predicate as composition of a function with its argument. This function checks whether the property of being on the road is among the properties listed in (4.30). This being said, it remains the case that quantifier expressions in natural language may behave differently from what you would expect after this training in predicate logic. A well-known peculiarity is that "a" may also be *generic*, with universal force. This would be written in logic with a universal quantifier:

A dog has fleas	$\forall x(Dx \to \exists y(Fy \wedge Hxy))$
A banker is trusted by no-one	$\forall x(Bx \to \forall y\neg Tyx)$.

It is quite puzzling that such generic force may result from the context in which an indefinite appears. Here is a famous example:

If a farmer owns a donkey he beats it. $\forall x \forall y((Fx \wedge Dy \wedge Oxy) \rightarrow Bxy)$.

Moving beyond philosophical disputes, a whole discipline of formal analysis of natural language has formed to attempt to solve such puzzles and create insight in how humans use language to convey meaning.

Summary of Things You Have Learnt in This Chapter *You have learnt a rich new logical language, which you can use to analyze mathematical but also natural language, as has been shown in many examples going from sentences to logical forms. You have learnt precisely how such a language gets a recursive compositional semantics in models with objects and predicates that can often be pictured very concretely. This is useful beyond this chapter, since this method has become a paradigm for formal semantics of languages in general. You have also acquired a first sense of validity and what is valid and invalid reasoning in this language. And finally, you have seen some glimpses of why this system is computationally more complex than what you ad learnt so far, illustrating the balance between expressive power and computational complexity that is the art underlying logic today.*

Further Reading Modern logic begins with the proposal for a formal language of pure thought by the German mathematician and philosopher Gottlob Frege. See [Fre67], in [Hei67]. Or if you read German: [Fre79]. Frege does not yet make a clear distinction between first order logic and higher order logic. The German mathematician David Hilbert seems to have been the first to stress this distinction. See the classic textbook [DA28] (in German), or its English translation [DA50]. Many excellent treatments of predicate logic exist, with emphasis on connections with philosophy [Hod01], with linguistics [Gam91], with mathematics [CH07], or with computer science [Bur98, HR04]. A comic style account of the development of modern logic can be found in [DP09] (also see `www.logicomix.com`).