

Atomic Sentences

In the Introduction, we talked about FOL as though it were a single language. Actually, it is more like a family of languages, all having a similar grammar and sharing certain important vocabulary items, known as the connectives and quantifiers. Languages in this family can differ, however, in the specific vocabulary used to form their most basic sentences, the so-called atomic sentences.

Atomic sentences correspond to the most simple sentences of English, sentences consisting of some names connected by a predicate. Examples are *Max ran*, *Max saw Claire*, and *Claire gave Scruffy to Max*. Similarly, in FOL atomic sentences are formed by combining names (or individual constants, as they are often called) and predicates, though the way they are combined is a bit different from English, as you will see.

atomic sentences

Different versions of FOL have available different names and predicates. We will frequently use a first-order language designed to describe blocks arranged on a chessboard, arrangements that you will be able to create in the program Tarski's World. This language has names like *b*, *e*, and *n₂*, and predicates like *Cube*, *Larger*, and *Between*. Some examples of atomic sentences in this language are *Cube(b)*, *Larger(c, f)*, and *Between(b, c, d)*. These sentences say, respectively, that *b* is a cube, that *c* is larger than *f*, and that *b* is between *c* and *d*.

names and predicates

Later in this chapter, we will look at the atomic sentences used in two other versions of FOL, the first-order languages of set theory and arithmetic. In the next chapter, we begin our discussion of the connectives and quantifiers common to all first-order languages.

SECTION 1.1

Individual constants

Individual constants are simply symbols that are used to refer to some fixed individual object. They are the FOL analogue of names, though in FOL we generally don't capitalize them. For example, we might use *max* as an individual constant to denote a particular person, named Max, or *1* as an individual constant to denote a particular number, the number one. In either case, they would basically work exactly the way names work in English. Our blocks

names in FOL

language takes the letters **a** through **f** plus n_1, n_2, \dots as its names.

The main difference between names in English and the individual constants of FOL is that we require the latter to refer to exactly one object. Obviously, the name *Max* in English can be used to refer to many different people, and might even be used twice in a single sentence to refer to two different people. Such wayward behavior is frowned upon in FOL.

There are also names in English that do not refer to any actually existing object. For example *Pegasus*, *Zeus*, and *Santa Claus* are perfectly fine names in English; they just fail to refer to anything or anybody. We don't allow such names in FOL.¹ What we do allow, though, is for one object to have more than one name; thus the individual constants **matthew** and **max** might both refer to the same individual. We also allow for nameless objects, objects that have no name at all.

Remember

In FOL,

- Every individual constant must name an (actually existing) object.
- No individual constant can name more than one object.
- An object can have more than one name, or no name at all.

SECTION 1.2

Predicate symbols

predicate or relation symbols

Predicate symbols are symbols used to express some property of objects or some relation between objects. Because of this, they are also sometimes called relation symbols. As in English, predicates are expressions that, when combined with names, form atomic sentences. But they don't correspond exactly to the predicates of English grammar.

logical subjects

Consider the English sentence *Max likes Claire*. In English grammar, this is analyzed as a subject-predicate sentence. It consists of the subject *Max* followed by the predicate *likes Claire*. In FOL, by contrast, we usually view this as a claim involving two “logical subjects,” the names *Max* and *Claire*, and

¹There is, however, a variant of first-order logic called *free logic* in which this assumption is relaxed. In free logic, there can be individual constants without referents. This yields a language more appropriate for mythology and fiction.

a predicate, *likes*, that expresses a relation between the referents of the names. Thus, atomic sentences of FOL often have two or more logical subjects, and the predicate is, so to speak, whatever is left. The logical subjects are called the “arguments” of the predicate. In this case, the predicate is said to be binary, since it takes two arguments.

*arguments of a
predicate*

In English, some predicates have optional arguments. Thus you can say *Claire gave*, *Claire gave Scruffy*, or *Claire gave Scruffy to Max*. Here the predicate *gave* is taking one, two, and three arguments, respectively. But in FOL, each predicate has a fixed number of arguments, a fixed *arity* as it is called. This is a number that tells you how many individual constants the predicate symbol needs in order to form a sentence. The term “arity” comes from the fact that predicates taking one argument are called *unary*, those taking two are *binary*, those taking three are *ternary*, and so forth.

arity of a predicate

If the arity of a predicate symbol *Pred* is 1, then *Pred* will be used to express some property of objects, and so will require exactly one argument (a name) to make a claim. For example, we might use the unary predicate symbol *Home* to express the property of being at home. We could then combine this with the name *max* to get the expression *Home(max)*, which expresses the claim that Max is at home.

If the arity of *Pred* is 2, then *Pred* will be used to represent a relation between two objects. Thus, we might use the expression *Taller(claire, max)* to express a claim about Max and Claire, the claim that Claire is taller than Max. In FOL, we can have predicate symbols of any arity. However, in the blocks language used in Tarski’s World we restrict ourselves to predicates with arities 1, 2, and 3. Here we list the predicates of that language, this time with their arity.

Arity 1: *Cube*, *Tet*, *Dodec*, *Small*, *Medium*, *Large*

Arity 2: *Smaller*, *Larger*, *LeftOf*, *RightOf*, *BackOf*, *FrontOf*, *SameSize*, *SameShape*, *SameRow*, *SameCol*, *Adjoins*, *=*

Arity 3: *Between*

Tarski’s World assigns each of these predicates a fixed interpretation, one reasonably consistent with the corresponding English verb phrase. For example, *Cube* corresponds to *is a cube*, *BackOf* corresponds to *is in back of*, and so forth. You can get the hang of them by working through the first set of exercises given below. To help you learn exactly what the predicates mean, Table 1.1 lists atomic sentences that use these predicates, together with their interpretations.

In English, predicates are sometimes vague. It is often unclear whether

vagueness

Table 1.1: Blocks language predicates.

Atomic Sentence	Interpretation
Tet(<i>a</i>)	<i>a</i> is a tetrahedron
Cube(<i>a</i>)	<i>a</i> is a cube
Dodec(<i>a</i>)	<i>a</i> is a dodecahedron
Small(<i>a</i>)	<i>a</i> is small
Medium(<i>a</i>)	<i>a</i> is medium
Large(<i>a</i>)	<i>a</i> is large
SameSize(<i>a</i> , <i>b</i>)	<i>a</i> is the same size as <i>b</i>
SameShape(<i>a</i> , <i>b</i>)	<i>a</i> is the same shape as <i>b</i>
Larger(<i>a</i> , <i>b</i>)	<i>a</i> is larger than <i>b</i>
Smaller(<i>a</i> , <i>b</i>)	<i>a</i> is smaller than <i>b</i>
SameCol(<i>a</i> , <i>b</i>)	<i>a</i> is in the same column as <i>b</i>
SameRow(<i>a</i> , <i>b</i>)	<i>a</i> is in the same row as <i>b</i>
Adjoins(<i>a</i> , <i>b</i>)	<i>a</i> and <i>b</i> are located on adjacent (but not diagonally) squares
LeftOf(<i>a</i> , <i>b</i>)	<i>a</i> is located nearer to the left edge of the grid than <i>b</i>
RightOf(<i>a</i> , <i>b</i>)	<i>a</i> is located nearer to the right edge of the grid than <i>b</i>
FrontOf(<i>a</i> , <i>b</i>)	<i>a</i> is located nearer to the front of the grid than <i>b</i>
BackOf(<i>a</i> , <i>b</i>)	<i>a</i> is located nearer to the back of the grid than <i>b</i>
Between(<i>a</i> , <i>b</i> , <i>c</i>)	<i>a</i> , <i>b</i> and <i>c</i> are in the same row, column, or diagonal, and <i>a</i> is between <i>b</i> and <i>c</i>

determinate property

an individual has the property in question or not. For example, Claire, who is sixteen, is young. She will not be young when she is 96. But there is no determinate age at which a person stops being young: it is a gradual sort of thing. FOL, however, assumes that every predicate is interpreted by a determinate property or relation. By a *determinate* property, we mean a property for which, given any object, there is a definite fact of the matter whether or not the object has the property.

This is one of the reasons we say that the blocks language predicates are

somewhat consistent with the corresponding English predicates. Unlike the English predicates, they are given very precise interpretations, interpretations that are suggested by, but not necessarily identical with, the meanings of the corresponding English phrases. The case where the discrepancy is probably the greatest is between **Between** and *is between*.

Remember

In FOL,

- Every predicate symbol comes with a single, fixed “arity,” a number that tells you how many names it needs to form an atomic sentence.
- Every predicate is interpreted by a determinate property or relation of the same arity as the predicate.

SECTION 1.3

Atomic sentences

In FOL, the simplest kinds of claims are those made with a single predicate and the appropriate number of individual constants. A sentence formed by a predicate followed by the right number of names is called an *atomic sentence*. For example **Taller**(*claire*, *max*) and **Cube**(*a*) are atomic sentences, provided the names and predicate symbols in question are part of the vocabulary of our language. In the case of the identity symbol, we put the two required names on either side of the predicate, as in **a** = **b**. This is called “infix” notation, since the predicate symbol = appears in between its two arguments. With the other predicates we use “prefix” notation: the predicate precedes the arguments.

atomic sentence

infix vs. prefix notation

The order of the names in an atomic sentence is quite important. Just as *Claire is taller than Max* means something different from *Max is taller than Claire*, so too **Taller**(*claire*, *max*) means something completely different than **Taller**(*max*, *claire*). We have set things up in our blocks language so that the order of the arguments of the predicates is like that in English. Thus **LeftOf**(*b*, *c*) means more or less the same thing as the English sentence ***b** is left of **c***, and **Between**(*b*, *c*, *d*) means roughly the same as the English ***b** is between **c** and **d***.

Predicates and names designate properties and objects, respectively. What

claims

truth value

makes sentences special is that they make claims (or express propositions). A claim is something that is either true or false; which of these it is we call its *truth value*. Thus `Taller(claire, max)` expresses a claim whose truth value is TRUE, while `Taller(max, claire)` expresses a claim whose truth value is FALSE. (You probably didn't know that, but now you do.) Given our assumption that predicates express determinate properties and that names denote definite individuals, it follows that each atomic sentence of FOL must express a claim that is either true or false.

You try it

- ▶ 1. It is time to try your hand using Tarski's World. In this exercise, you will use Tarski's World to become familiar with the interpretations of the atomic sentences of the blocks language. Before starting, though, you need to learn how to launch Tarski's World and perform some basic operations. Read the appropriate sections of the user's manual describing Tarski's World before going on.
- ▶ 2. Launch Tarski's World and open the files called **Wittgenstein's World** and **Wittgenstein's Sentences**. You will find these in the folder **TW Exercises**. In these files, you will see a blocks world and a list of atomic sentences. (We have added comments to some of the sentences. Comments are prefaced by a semicolon (";"), which tells Tarski's World to ignore the rest of the line.)
- ▶ 3. Move through the sentences using the arrow keys on your keyboard, mentally assessing the truth value of each sentence in the given world. Use the **Verify Sentence** button to check your assessments. This button is on the left of the group of three colored buttons on the toolbar (the one which has **T/F** written on it). (Since the sentences are all atomic sentences the **Game** button, on the right of the same group, will not be helpful.) If you are surprised by any of the evaluations, try to figure out how your interpretation of the predicate differs from the correct interpretation.
- ▶ 4. Next change **Wittgenstein's World** in many different ways, seeing what happens to the truth of the various sentences. The main point of this is to help you figure out how Tarski's World interprets the various predicates. For example, what does `BackOf(d, c)` mean? Do two things have to be in the same column for one to be in back of the other?
- ▶ 5. Play around as much as you need until you are sure you understand the meanings of the atomic sentences in this file. For example, in the original

world none of the sentences using **Adjoins** comes out true. You should try to modify the world to make some of them true. As you do this, you will notice that large blocks cannot adjoin other blocks.

6. In doing this exercise, you will no doubt notice that **Between** does not mean exactly what the English *between* means. This is due to the necessity of interpreting **Between** as a determinate predicate. For simplicity, we insist that in order for b to be between c and d , all three must be in the same row, column, or diagonal. ◀
7. When you are finished, close the files, but do not save the changes you have made to them. ◀

..... *Congratulations*

Remember

In FOL,

- Atomic sentences are formed by putting a predicate of arity n in front of n names (enclosed in parentheses and separated by commas).
- Atomic sentences are built from the identity predicate, $=$, using infix notation: the arguments are placed on either side of the predicate.
- The order of the names is crucial in forming atomic sentences.

Exercises

You will eventually want to read the entire chapter of the user's manual on how to use Tarski's World. To do the following problems, you will need to read at least the first four sections. Also, if you don't remember how to name and submit your solution files, you should review the section on essential instructions in the Introduction, starting on page 5.

- 1.1 If you skipped the **You try it** section, go back and do it now. This is an easy but crucial exercise that will familiarize you with the atomic sentences of the blocks language. There is nothing you need to turn in or submit, but don't skip the exercise!
- 1.2 (Copying some atomic sentences) This exercise will give you some practice with the Tarski's World keyboard window, as well as with the syntax of atomic sentences. The following are all atomic sentences of our language. Start a new sentence file and copy them into it. Have Tarski's World check each formula after you write it to see that it is a sentence. If you make a mistake, edit it before going on. Make sure you use the **Add Sentence** command between sentences, ↗

not the return key. If you've done this correctly, the sentences in your list will be numbered and separated by horizontal lines.

1. Tet(a)
2. Medium(a)
3. Dodec(b)
4. Cube(c)
5. FrontOf(a, b)
6. Between(a, b, c)
7. a = d
8. Larger(a, b)
9. Smaller(a, c)
10. LeftOf(b, c)

Remember, you should save these sentences in a file named **Sentences 1.2**. When you've finished your first assignment, submit all of your solution files using the Submit program.

1.3 (Building a world) Build a world in which all the sentences in Exercise 1.2 are simultaneously true. Remember to name and submit your world file as **World 1.3**.

1.4 (Translating atomic sentences) Here are some simple sentences of English. Start a new sentence file and translate them into FOL.

1. *a is a cube.*
2. *b is smaller than a.*
3. *c is between a and d.*
4. *d is large.*
5. *e is larger than a.*
6. *b is a tetrahedron.*
7. *e is a dodecahedron.*
8. *e is right of b.*
9. *a is smaller than e.*
10. *d is in back of a.*
11. *b is in the same row as d.*
12. *b is the same size as c.*

After you've translated the sentences, build a world in which all of your translations are true. Submit your sentence and world files as **Sentences 1.4** and **World 1.4**.

1.5 (Naming objects) Open Lestrade's Sentences and Lestrade's World. You will notice that none of the objects in this world has a name. Your task is to assign the objects names in such a way that all the sentences in the list come out true. Remember to save your solution in a file named **World 1.5**. Be sure to use **Save World As...**, not **Save World**.

1.6



(Naming objects, continued) Not all of the choices in Exercise 1.5 were forced on you. That is, you could have assigned the names differently and still had the sentences come out true. Change the assignment of as many names as possible while still making all the sentences true, and submit the changed world as **World 1.6**. In order for us to compare your files, you must submit both **World 1.5** and **World 1.6** at the same time.

1.7



(Context sensitivity of predicates) We have stressed the fact that FOL assumes that every predicate is interpreted by a determinate relation, whereas this is not the case in natural languages like English. Indeed, even when things seem quite determinate, there is often some form of context sensitivity. In fact, we have built some of this into Tarski's World. Consider, for example, the difference between the predicates **Larger** and **BackOf**. Whether or not cube a is larger than cube b is a determinate matter, and also one that does not vary depending on your perspective on the world. Whether or not a is back of b is also determinate, but in this case it does depend on your perspective. If you rotate the world by 90° , the answer might change.

Open **Austin's Sentences** and **Wittgenstein's World**. Evaluate the sentences in this file and tabulate the resulting truth values in a table like the one below. We've already filled in the first column, showing the values in the original world. Rotate the world 90° clockwise and evaluate the sentences again, adding the results to the table. Repeat until the world has come full circle.

	Original	Rotated 90°	Rotated 180°	Rotated 270°
1.	FALSE			
2.	FALSE			
3.	TRUE			
4.	FALSE			
5.	TRUE			
6.	FALSE			

You should be able to think of an atomic sentence in the blocks language that would produce a row across the table with the following pattern:

TRUE FALSE TRUE FALSE

Add a seventh sentence to **Austin's Sentences** that would display the above pattern.

Are there any atomic sentences in the language that would produce a row with this pattern?

FALSE TRUE FALSE FALSE

If so, add such a sentence as sentence eight in **Austin's Sentences**. If not, leave sentence eight blank.

Are there any atomic sentences that would produce a row in the table containing exactly three TRUE's? If so, add such a sentence as number nine. If not, leave sentence nine blank.

Submit your modified sentence file as **Sentences 1.7**. Turn in your completed table to your instructor.

SECTION 1.4

General first-order languages

First-order languages differ in the names and predicates they contain, and so in the atomic sentences that can be formed. What they share are the connectives and quantifiers that enable us to build more complex sentences from these simpler parts. We will get to those common elements in later chapters.

translation

When you translate a sentence of English into FOL, you will sometimes have a “predefined” first-order language that you want to use, like the blocks language of Tarski’s World, or the language of set theory or arithmetic described later in this chapter. If so, your goal is to come up with a translation that captures the meaning of the original English sentence as nearly as possible, given the names and predicates available in your predefined first-order language.

designing languages

Other times, though, you will not have a predefined language to use for your translation. If not, the first thing you have to do is decide what names and predicates you need for your translation. In effect, you are designing, on the fly, a new first-order language capable of expressing the English sentence you want to translate. We’ve been doing this all along, for example when we introduced $\text{Home}(\text{max})$ as the translation of *Max is at home* and $\text{Taller}(\text{claire}, \text{max})$ as the translation of *Claire is taller than Max*.

When you make these decisions, there are often alternative ways to go. For example, suppose you were asked to translate the sentence *Claire gave Scruffy to Max*. You might introduce a binary predicate $\text{GaveScruffy}(x, y)$, meaning *x gave Scruffy to y*, and then translate the original sentence as $\text{GaveScruffy}(\text{claire}, \text{max})$. Alternatively, you might introduce a three-place predicate $\text{Gave}(x, y, z)$, meaning *x gave y to z*, and then translate the sentence as $\text{Gave}(\text{claire}, \text{scruffy}, \text{max})$.

choosing predicates

There is nothing wrong with either of these predicates, or their resulting translations, so long as you have clearly specified what the predicates mean. Of course, they may not be equally useful when you go on to translate other sentences. The first predicate will allow you to translate sentences like *Max gave Scruffy to Evan* and *Evan gave Scruffy to Miles*. But if you then run into the sentence *Max gave Carl to Claire*, you would be stuck, and would have to introduce an entirely new predicate, say, $\text{GaveCarl}(x, y)$. The three-place predicate is thus more flexible. A first-order language that contained it (plus the relevant names) would be able to translate any of these sentences.

In general, when designing a first-order language we try to economize on the predicates by introducing more flexible ones, like $\text{Gave}(x, y, z)$, rather than

less flexible ones, like $\text{GaveScruffy}(x, y)$ and $\text{GaveCarl}(x, y)$. This produces a more expressive language, and one that makes the logical relations between various claims more perspicuous.

Names can be introduced into a first-order language to refer to anything that can be considered an object. But we construe the notion of an “object” pretty flexibly—to cover anything that we can make claims about. We’ve already seen languages with names for people and the blocks of Tarski’s World. Later in the chapter, we’ll introduce languages with names for sets and numbers. Sometimes we will want to have names for still other kinds of “objects,” like days or times. Suppose, for example, that we want to translate the sentences:

objects

*Claire gave Scruffy to Max on Saturday.
Sunday, Max gave Scruffy to Evan.*

Here, we might introduce a four-place predicate $\text{Gave}(w, x, y, z)$, meaning *w gave x to y on day z*, plus names for particular days, like last Saturday and last Sunday. The resulting translations would look something like this:

$\text{Gave}(\text{claire}, \text{scruffy}, \text{max}, \text{saturday})$
 $\text{Gave}(\text{max}, \text{scruffy}, \text{evan}, \text{sunday})$

Designing a first-order language with just the right names and predicates requires some skill. Usually, the overall goal is to come up with a language that can say everything you want, but that uses the smallest “vocabulary” possible. Picking the right names and predicates is the key to doing this.

Exercises

1.8



Suppose we have two first-order languages: the first contains the binary predicates $\text{GaveScruffy}(x, y)$ and $\text{GaveCarl}(x, y)$, and the names *max* and *claire*; the second contains the ternary predicate $\text{Gave}(x, y, z)$ and the names *max*, *claire*, *scruffy*, and *carl*.

1. List all of the atomic sentences that can be expressed in the first language. (Some of these may say weird things like $\text{GaveScruffy}(\text{claire}, \text{claire})$, but don’t worry about that.)
2. How many atomic sentences can be expressed in the second language? (Count all of them, including odd ones like $\text{Gave}(\text{scruffy}, \text{scruffy}, \text{scruffy})$.)
3. How many names and binary predicates would a language like the first need in order to say everything you can say in the second?

Table 1.2: Names and predicates for a language.

ENGLISH	FOL	COMMENT
Names:		
<i>Max</i>	max	
<i>Claire</i>	claire	
<i>Folly</i>	folly	The name of a certain dog.
<i>Carl</i>	carl	The name of another dog.
<i>Scruffy</i>	scruffy	The name of a certain cat.
<i>Pris</i>	pris	The name of another cat.
<i>2 pm, Jan 2, 2011</i>	2:00	The name of a time.
<i>2:01 pm, Jan 2, 2011</i>	2:01	One minute later.
\vdots	\vdots	Similarly for other times.
Predicates:		
<i>x is a pet</i>	Pet(x)	
<i>x is a person</i>	Person(x)	
<i>x is a student</i>	Student(x)	
<i>x is at home</i>	Home(x)	
<i>x is happy</i>	Happy(x)	
<i>t is earlier than t'</i>	$t < t'$	Earlier-than for times.
<i>x was hungry at time t</i>	Hungry(x, t)	
<i>x was angry at time t</i>	Angry(x, t)	
<i>x owned y at time t</i>	Owned(x, y, t)	
<i>x gave y to z at t</i>	Gave(x, y, z, t)	
<i>x fed y at time t</i>	Fed(x, y, t)	

1.9



We will be giving a number of problems that use the symbols explained in Table 1.2. Start a new sentence file in Tarski's World and translate the following into FOL, using the names and predicates listed in the table. (You can switch to the Pets language in the Sentence toolbar. When you type, make sure they appear exactly as in the table; for example, use 2:00, not 2:00 pm or 2 pm.) All references to times are assumed to be to times on January 2, 2011.

1. *Claire owned Folly at 2 pm.*
2. *Claire gave Pris to Max at 2:05 pm.*
3. *Max is a student.*
4. *Claire fed Carl at 2 pm.*
5. *Folly belonged to Max at 3:05 pm.*
6. *2:00 pm is earlier than 2:05 pm.*

Name and submit your file in the usual way.

1.10 Translate the following into natural sounding, colloquial English, consulting Table 1.2.



1. Owned(max, scruffy, 2:00)
2. Fed(max, scruffy, 2:30)
3. Gave(max, scruffy, claire, 3:00)
4. 2:00 < 2:00

1.11 For each sentence in the following list, suggest a translation into an atomic sentence of FOL. In addition to giving the translation, explain what kinds of objects your names refer to and the intended meaning of the predicate you use.



1. *Max shook hands with Claire.*
2. *Max shook hands with Claire yesterday.*
3. *AIDS is less contagious than influenza.*
4. *Spain is between France and Portugal in size.*
5. *Misery loves company.*

SECTION 1.5

Function symbols

Some first-order languages have, in addition to names and predicates, other expressions that can appear in atomic sentences. These expressions are called *function symbols*. Function symbols allow us to form name-like terms from names and other name-like terms. They allow us to express, using atomic sentences, complex claims that could not be perspicuously expressed using just names and predicates. Some English examples will help clarify this.

function symbols

English has many sorts of noun phrases, expressions that can be combined with a verb phrase to get a sentence. Besides names like *Max* and *Claire*, other noun phrases include expressions like *Max's father*, *Claire's mother*, *Every girl who knows Max*, *No boy who knows Claire*, *Someone* and so forth. Each of these combines with a singular verb phrase such as *likes unbuttered popcorn* to make a sentence. But notice that the sentences that result have very different logical properties. For example,

Claire's mother likes unbuttered popcorn

implies that someone likes unbuttered popcorn, while

No boy who knows Claire likes unbuttered popcorn

does not.

Since these noun phrases have such different logical properties, they are treated differently in FOL. Those that intuitively refer to an individual are

terms

called “terms,” and behave like the individual constants we have already discussed. In fact, individual constants are the simplest terms, and more complex terms are built from them using function symbols. Noun phrases like *No boy who knows Claire* are handled with very different devices, known as quantifiers, which we will discuss later.

complex terms

The FOL analog of the noun phrase *Max’s father* is the term `father(max)`. It is formed by putting a function symbol, `father`, in front of the individual constant `max`. The result is a complex term that we use to refer to the father of the person referred to by the name `max`. Similarly, we can put the function symbol `mother` together with the name `claire` and get the term `mother(claire)`, which functions pretty much like the English term *Claire’s mother*.

We can repeat this construction as many times as we like, forming more and more complex terms:

$$\begin{aligned} &\text{father}(\text{father}(\text{max})) \\ &\text{mother}(\text{father}(\text{claire})) \\ &\text{mother}(\text{mother}(\text{mother}(\text{claire}))) \end{aligned}$$

The first of these refers to Max’s paternal grandfather, the second to Claire’s paternal grandmother, and so forth.

These function symbols are called unary function symbols, because, like unary predicates, they take one argument. The resulting terms function just like names, and can be used in forming atomic sentences. For instance, the FOL sentence

$$\text{Taller}(\text{father}(\text{max}), \text{max})$$

says that Max’s father is taller than Max. Thus, in a language containing function symbols, the definition of atomic sentence needs to be modified to allow complex terms to appear in the argument positions in addition to names.

*function symbols vs.
predicates*

Students often confuse function symbols with predicates, because both take terms as arguments. But there is a big difference. When you combine a unary function symbol with a term you do not get a sentence, but another term: something that refers (or should refer) to an object of some sort. This is why function symbols can be reapplied over and over again. As we have seen, the following makes perfectly good sense:

$$\text{father}(\text{father}(\text{max}))$$

This, on the other hand, is total nonsense:

$$\text{Dodec}(\text{Dodec}(\text{a}))$$

To help prevent this confusion, we will always capitalize predicates of FOL and leave function symbols and names in lower case.

Besides unary function symbols, FOL allows function symbols of any arity. Thus, for example, we can have binary function symbols. Simple English counterparts of binary function symbols are hard to come up with, but they are quite common in mathematics. For instance, we might have a function symbol `sum` that combines with two terms, t_1 and t_2 , to give a new term, `sum(t_1, t_2)`, which refers to the sum of the numbers referred to by t_1 and t_2 . Then the complex term `sum(3, 5)` would give us another way of referring to 8. In a later section, we will introduce a function symbol to denote addition, but we will use infix notation, rather than prefix notation. Thus $3 + 5$ will be used instead of `sum(3, 5)`.

*arity of function
symbols*

In FOL, just as we assume that every name refers to an actual object, we also assume that every complex term refers to exactly one object. This is a somewhat artificial assumption, since many function-like expressions in English don't always work this way. Though we may assume that

`mother(father(father(max)))`

refers to an actual (deceased) individual—one of Max's great-grandmothers—there may be other uses of these function symbols that don't seem to give us genuinely referring expressions. For example, perhaps the complex terms `mother(adam)` and `mother(eve)` fail to refer to any individuals, if Adam and Eve were in fact the first people. And certainly the complex term `mother(3)` doesn't refer to anything, since the number three has no mother. When designing a first-order language with function symbols, you should try to ensure that your complex terms always refer to unique, existing individuals.

The blocks world language as it is implemented in Tarski's World does not contain function symbols, but we could easily extend the language to include some. Suppose for example we introduced the function expressions `fm`, `bm`, `lm` and `rm`, that allowed us to form complex terms like:

*functions symbols for
blocks language*

`fm(a)`
`lm(bm(c))`
`rm(rm(fm(d)))`

We could interpret these function symbols so that, for example, `fm(a)` refers to the frontmost block in the same column as a . Thus, if there are several blocks in the column with a , then `fm(a)` refers to whichever one is nearest the front. (Notice that `fm(a)` may not itself have a name; `fm(a)` may be our only way to refer to it.) If a is the only block in the column, or is the frontmost in its column, then `fm(a)` would refer to a . Analogously, `bm`, `lm` and `rm` could be interpreted to mean *backmost*, *leftmost* and *rightmost*, respectively.

With this interpretation, the term `lm(bm(c))` would refer to the leftmost block in the same row as the backmost block in the same column as c . The

atomic sentence $\text{Larger}(\text{lm}(\text{bm}(c)), c)$ would then be true if and only if this block is larger than c .

Notice that in this expanded language, the sentence $\text{lm}(\text{bm}(c)) = \text{bm}(\text{lm}(c))$ is not always true. (Can you think of an example where it is false?) On the other hand, $\text{fm}(\text{fm}(a)) = \text{fm}(a)$ is always true. Can you think of any other atomic sentences using these function symbols that are always true? How about sentences that are always false?

Remember

In a language with function symbols,

- Complex terms are typically formed by putting a function symbol of arity n in front of n terms (simple or complex).
- Complex terms are used just like names (simple terms) in forming atomic sentences.
- In FOL, complex terms are assumed to refer to one and only one object.

Exercises

- 1.12** Express in English the claims made by the following sentences of FOL as clearly as you can. You should try to make your English sentences as natural as possible. All the sentences are, by the way, true.



1. $\text{Taller}(\text{father}(\text{claire}), \text{father}(\text{max}))$
2. $\text{john} = \text{father}(\text{max})$
3. $\text{Taller}(\text{claire}, \text{mother}(\text{mother}(\text{claire})))$
4. $\text{Taller}(\text{mother}(\text{mother}(\text{max})), \text{mother}(\text{father}(\text{max})))$
5. $\text{mother}(\text{melanie}) = \text{mother}(\text{claire})$

- 1.13** Assume that we have expanded the blocks language to include the function symbols fm , bm , lm and rm described earlier. Then the following formulas would all be sentences of the language:





1. $\text{Tet}(\text{lm}(e))$
2. $\text{fm}(c) = c$
3. $\text{bm}(b) = \text{bm}(e)$
4. $\text{FrontOf}(\text{fm}(e), e)$
5. $\text{LeftOf}(\text{fm}(b), b)$

6. SameRow(rm(c), c)
7. $\text{bm}(\text{lm}(c)) = \text{lm}(\text{bm}(c))$
8. SameShape(lm(b), bm(rm(e)))
9. $d = \text{lm}(\text{fm}(\text{rm}(\text{bm}(d))))$
10. Between(b, lm(b), rm(b))

Fill in the following table with TRUE's and FALSE's according to whether the indicated sentence is true or false in the indicated world. Since Tarski's World does not understand the function symbols, you will not be able to check your answers. We have filled in a few of the entries for you. Turn in the completed table to your instructor.

	Leibniz's	Bolzano's	Boole's	Wittgenstein's
1.	TRUE			
2.				
3.				
4.				
5.	FALSE			
6.		TRUE		
7.				
8.			FALSE	
9.				
10.				

1.14  As you probably noticed in doing Exercise 1.13, three of the sentences came out true in all four worlds. It turns out that one of these three cannot be falsified in any world, because of the meanings of the predicates and function symbols it contains. Your goal in this problem is to build a world in which all of the other sentences in Exercise 1.13 come out false. When you have found such a world, submit it as World 1.14.

1.15  Suppose we have two first-order languages for talking about fathers. The first, which we'll call the functional language, contains the names *claire*, *melanie*, and *jon*, the function symbol *father*, and the predicates = and *Taller*. The second language, which we will call the relational language, has the same names, no function symbols, and the binary predicates =, *Taller*, and *FatherOf*, where *FatherOf*(*c*, *b*) means that *c* is the father of *b*. Translate the following atomic sentences from the relational language into the functional language. Be careful. Some atomic sentences, such as *claire* = *claire*, are in both languages! Such a sentence counts as a translation of itself.

1. *FatherOf*(jon, claire)
2. *FatherOf*(jon, melanie)

3. $\text{Taller}(\text{claire}, \text{melanie})$

Which of the following atomic sentences of the functional language can be translated into atomic sentences of the relational language? Translate those that can be and explain the problem with those that can't.

4. $\text{father}(\text{melanie}) = \text{jon}$
5. $\text{father}(\text{melanie}) = \text{father}(\text{claire})$
6. $\text{Taller}(\text{father}(\text{claire}), \text{father}(\text{jon}))$

When we add connectives and quantifiers to the language, we will be able to translate freely back and forth between the functional and relational languages.

1.16



Let's suppose that everyone has a favorite movie star. Given this assumption, make up a first-order language for talking about people and their favorite movie stars. Use a function symbol that allows you to refer to an individual's favorite actor, plus a relation symbol that allows you to say that one person is a better actor than another. Explain the interpretation of your function and relation symbols, and then use your language to express the following claims:

1. *Harrison is Nancy's favorite actor.*
2. *Nancy's favorite actor is better than Sean.*
3. *Nancy's favorite actor is better than Max's.*
4. *Claire's favorite actor's favorite actor is Brad.*
5. *Sean is his own favorite actor.*

1.17



Make up a first-order language for talking about people and their relative heights. Instead of using relation symbols like **Taller**, however, use a function symbol that allows you to refer to people's heights, plus the relation symbols $=$ and $<$. Explain the interpretation of your function symbol, and then use your language to express the following two claims:

1. *George is taller than Sam.*
2. *Sam and Mary are the same height.*

Do you see any problem with this function symbol? If so, explain the problem. [Hint: What happens if you apply the function symbol twice?]

1.18



For each sentence in the following list, suggest a translation into an atomic sentence of FOL. In addition to giving the translation, explain what kinds of objects your names refer to and the intended meaning of the predicates and function symbols you use.

1. *Indiana's capital is larger than California's.*
2. *Hitler's mistress died in 1945.*
3. *Max shook Claire's father's hand.*
4. *Max is his father's son.*
5. *John and Nancy's eldest child is younger than Jon and Mary Ellen's.*

The first-order language of set theory

FOL was initially developed for use in mathematics, and consequently the most familiar first-order languages are those associated with various branches of mathematics. One of the most common of these is the language of set theory. This language has only two predicates, both binary. The first is the identity symbol, $=$, which we have already encountered, and the second is the symbol \in , for set membership.

predicates of set theory

It is standard to use infix notation for both of these predicates. Thus, in set theory, atomic sentences are always formed by placing individual constants on either side of one of the two predicates. This allows us to make identity claims, of the form $a = b$, and membership claims, of the form $a \in b$ (where a and b are individual constants).

A sentence of the form $a \in b$ is true if and only if the thing named by b is a set, and the thing named by a is a member of that set. For example, suppose a names the number 2 and b names the set $\{2, 4, 6\}$. Then the following table tells us which membership claims made up using these names are true and which are false.²

membership (\in)

$a \in a$	FALSE
$a \in b$	TRUE
$b \in a$	FALSE
$b \in b$	FALSE

Notice that there is one striking difference between the atomic sentences of set theory and the atomic sentences of the blocks language. In the blocks language, you can have a sentence, like $\text{LeftOf}(a, b)$, that is true in a world, but which can be made false simply by moving one of the objects. Moving an object does not change the way the name works, but it can turn a true sentence into a false one, just as the sentence *Claire is sitting down* can go from true to false in virtue of Claire's standing up.

In set theory, we won't find this sort of thing happening. Here, the analog of a world is just a domain of objects and sets. For example, our domain might consist of all natural numbers, sets of natural numbers, sets of sets of natural numbers, and so forth. The difference between these "worlds" and those of Tarski's World is that the truth or falsity of the atomic sentences is determined entirely once the reference of the names is fixed. There is nothing that corresponds to moving the blocks around. Thus if the universe contains

²For the purposes of this discussion we are assuming that numbers are not sets, and that sets can contain either numbers or other sets as members.

the objects 2 and $\{2, 4, 6\}$, and if the names **a** and **b** are assigned to them, then the atomic sentences must get the values indicated in the previous table. The only way those values can change is if the names name different things. Identity claims also work this way, both in set theory and in Tarski's World.

Exercises

1.19 Which of the following atomic sentences in the first-order language of set theory are true and which are false? We use, in addition to **a** and **b** as above, the name **c** for 6 and **d** for $\{2, 7, \{2, 4, 6\}\}$.



1. $a \in c$
2. $a \in d$
3. $b \in c$
4. $b \in d$
5. $c \in d$
6. $c \in b$

To answer this exercise, submit a Tarski's World sentence file with an uppercase **T** or **F** in each sentence slot to indicate your assessment.

SECTION 1.7

The first-order language of arithmetic

While neither the blocks language as implemented in Tarski's World nor the language of set theory has function symbols, there are languages that use them extensively. One such first-order language is the language of arithmetic. This language allows us to express statements about the natural numbers $0, 1, 2, 3, \dots$, and the usual operations of addition and multiplication.

*predicates ($=, <$) and
functions ($+, \times$) of
arithmetic*

There are several more or less equivalent ways of setting up this language. The one we will use has two names, **0** and **1**, two binary relation symbols, $=$ and $<$, and two binary function symbols, $+$ and \times . The atomic sentences are those that can be built up out of these symbols. We will use infix notation both for the relation symbols and the function symbols.

Notice that there are infinitely many different terms in this language (for example, **0**, **1**, $(1 + 1)$, $((1 + 1) + 1)$, $((((1 + 1) + 1) + 1) + 1)$, \dots), and so an infinite number of atomic sentences. Our list also shows that every natural number is named by some term of the language. This raises the question of how we can specify the set of terms in a precise way. We can't list them all explicitly, since

there are too many. The way we get around this is by using what is known as an *inductive* definition.

Definition The terms of first-order arithmetic are formed in the following way:

terms of arithmetic

1. The names $0, 1$ are terms.
2. If t_1, t_2 are terms, then the expressions $(t_1 + t_2)$ and $(t_1 \times t_2)$ are also terms.
3. Nothing is a term unless it can be obtained by repeated application of (1) and (2).

We should point out that this definition does indeed allow the function symbols to be applied over and over. Thus, $(1 + 1)$ is a term by clause 2 and the fact that 1 is a term. In which case $((1 + 1) \times (1 + 1))$ is also a term, again by clause 2. And so forth.


The third clause in the above definition is not as straightforward as one might want, since the phrase “can be obtained by repeated application of” is a bit vague. In Chapter 16, we will see how to give definitions like the above in a more satisfactory way, one that avoids this vague clause.

The atomic sentences in the language of first-order arithmetic are those that can be formed from the terms and the two binary predicate symbols, $=$ and $<$. So, for example, the FOL version of *1 times 1 is less than 1 plus 1* is the following:


atomic sentences of arithmetic


$$(1 \times 1) < (1 + 1)$$

Exercises

1.20  Show that the following expressions are terms in the first-order language of arithmetic. Do this by explaining which clauses of the definition are applied and in what order. What numbers do they refer to?

1. $(0 + 0)$
2. $(0 + (1 \times 0))$
3. $((1 + 1) + ((1 + 1) \times (1 + 1)))$
4. $((((1 \times 1) \times 1) \times 1))$

1.21  Find a way to express the fact that three is less than four using the first-order language of arithmetic.

1.22  Show that there are infinitely many terms in the first-order language of arithmetic referring to the number one.

SECTION 1.8

Alternative notation

As we said before, FOL is like a family of languages. But, as if that were not enough diversity, even the very same first-order language comes in a variety of dialects. Indeed, almost no two logic books use exactly the same notational conventions in writing first-order sentences. For this reason, it is important to have some familiarity with the different dialects—the different notational conventions—and to be able to translate smoothly between them. At the end of most chapters, we discuss common notational differences that you are likely to encounter.

Some notational differences, though not many, occur even at the level of atomic sentences. For example, some authors insist on putting parentheses around atomic sentences whose binary predicates are in infix position. So $(a = b)$ is used rather than $a = b$. By contrast, some authors omit parentheses surrounding the argument positions (and the commas between them) when the predicate is in prefix position. These authors use Rab instead of $R(a, b)$. We have opted for the latter simply because we use predicates made up of several letters, and the parentheses make it clear where the predicate ends and the arguments begin: $Cubed$ is not nearly as perspicuous as $Cube(d)$.

What is important in these choices is that sentences should be unambiguous and easy to read. Typically, the first aim requires parentheses to be used in one way or another, while the second suggests using no more than is necessary.