Chapter 4

PL—A Better Theory of Logical Form

The theory, SL, that we have studied so far can be put to serious use. However, our interest in it has been mainly as an example to illustrate the notion of theories of logical form. When serious work requiring logical theory is done, SL is likely to be too weak a theory to be entirely successful. Here are three general respects in which we might have reason to be dissatisfied with SL, and motivated to look for a more powerful theory of logical form.

Many common valid arguments can only have their validity explained in terms of features that are ignored by SL. In other words, many valid arguments are not solely valid due to the truth functional structure that they have, and so are not revealed to be valid by using SL. When an argument is not SL valid (valid due to its form according to SL), that may be because the argument is not valid at all, but it may also be because the argument is valid after all, but for reasons that SL does not recognize. For example, the valid argument,

No triangle is a rectangle
Some quadrilaterals are rectangles
Some quadrilaterals are not triangles

cannot be shown valid using SL. Since the vast majority of interesting arguments in mathematics, for instance, involves logical structure similar to that in the example, SL has serious deficiencies.

A related problem with SL, especially for the analysis of reasoning in mathematics and related areas, is that SL has no way of dealing with the most common sort of mathematical reasoning, involving information about equality (or equations).

A third problem with SL is that it is completely unsuited for another sort of activity important in mathematics and other areas that involve precise reasoning and proof, namely definition. We will see that by extending SL we will be able to understand definition as an application of logical analysis.

So, we will now present an improved theory of logical form, PL (for “Predicate Logic”). We start by introducing the catalogue, statement forms, and logical analysis of PL. Since each of these components of the theory is more complicated than the similar portion of SL, we will come back to them for a second look later on.
I. The Catalogue

The catalogue of PL has everything from SL plus additional items. The additional items are of two sorts. A new kind of logical operator, quantifiers, is added to the familiar truth functions. Also, PL concerns itself with the structure of basic statements containing no logical operators (non-sentence components of sentences). We introduce two “logical parts of speech”, terms and predicates in addition to the logical operators and statements already recognized in SL. In this first look at the catalogue, we will concentrate on terms and predicates.

Terms

Terms are in many ways the logician’s version of what grammarians call (singular) noun phrases. This is not a definition, but just a ball park idea of what terms are. There are two sorts of terms in PL, designators (also called “constants”) and variables. For now we will discuss designators.

Designators

Designators are noun phrases that are used in the way that (singular) proper names are typically used — to pick out, name, refer to, or describe some one particular thing. For instance, in the sentence

"Jack" went up the hill

"Jack" is used as a designator to pick out who is being said to have gone up the hill. We may not know who the person making the statement has in mind, but we do know that they are saying that someone that they have in mind went up the hill. We expect that it should be possible for someone making the statement to answer the question: Who do you mean to refer to by “Jack?”

Proper names are one example of designators, but not the only example. Pronouns often function as designators too. For example, “he” in

He went up the hill

might refer to Jack (or some other specific person). In that case, because it functions in the special way that designators function, “he” would be a designator in the sentence. Descriptions can also function as designators. For example, the underlined portion of

The student with the highest average went up the hill

is used as a designator.

On the other hand, when a noun phrase refers to more than one thing, the phrase is not a designator. Example: the entire underlined phrase is not a designator

Jack and Jill went up the hill.
However, that phrase contains two designators within it,

Jack
Jill.

Another case to consider:

No one went up the hill.

In this case, the underlined material is not a designator either. We do not even expect an answer to the (very weird) question. To whom were you referring when you said, “no one?”

Designators can refer to people, places, tables, books, cities, planets, and even numbers. Logicians are very liberal in what they count as a “thing.”

Predicates

Predicates are labeled, functional, open sentences. Now all I must do is tell you what an open sentence is, what a functional open sentence is, and what a labeled open sentence is.

Open Sentences

An open sentence is nothing more than what you get by starting with a sentence, removing any number of designators that you like and marking the spots that used to have designators with dashes. For example, starting with the sentence,

Jack went up the hill

we can get the open sentence,

_____ went up the hill.

Starting with the sentence,

Jack likes Jill

we can get these open sentences,

_____ likes Jill

Jack likes _____

_____ likes ____.

Labeled Open Sentences

We will find it useful to be able easily to refer to the blanks in an open sentence (and also to group some blanks together). That is the purpose of “labelling” the blanks in an open sentence. Labels are any suitable collection of symbols that are not used for other purposes in our notation. To begin with (though not forever) we will use numerals in boxes as labels. For example, [1], [2], and [3] are boxed numerals that we will use to label blanks in open sentences. Here are some examples of labelling blanks in open sentences:
Notice that it is ok for the numerals in the boxes to be in any order, and that it is ok for the same label to be used several times.

We will find it useful to count the number of different labels in use in an open sentence. We will call that number the “arity” or “number of places” in the open sentence. So the open sentences just above are, respectively,

1-place 2-place 2-place 1-place Attention Here!

open sentences. We will count two open sentences as being the same, if they use words with the same meaning and have their blanks labeled in the same way. So, in the example above, all four open sentences are counted as different open sentences from the others. The first one is different because of a difference in the meaning of the words. The others are different from each other because of the difference in labelling.

Functional Open Sentences

Not all open sentences are alike. There is a special kind of open sentence that is of great interest to logicians. In a rather abstract way, the difference between the interesting and the uninteresting open sentences is similar to the difference between the interesting statement constructions (truth functional constructions) and the uninteresting statement constructions. We will first illustrate the two different types of open sentences and then give a definition of the interesting type. Consider these two open sentences:

___ is a wrestler
___ is so-called because of his size.

Now consider these two designators:

Steven Hogan
Hulk Hogan.

In case you don’t follow professional wrestling, those two designators refer to the very same person.

Now form sentences by filling in the blank in the first of our two open sentences with one of the designators and then with the other:

Steven Hogan is a wrestler
Hulk Hogan is a wrestler.

Both of these sentences are true (as it happens in this case). But even if you didn’t know that Hogan is a wrestler,
once I tell you that the two designators refer to the very same person, you expect that the two sentences will have the same truth value, both true or both false. Do the same thing with the other open sentence:

Steven Hogan is so-called because of his size
Hulk Hogan is so-called because of his size.

In this case, the two sentences differ in truth value. This first is false and the second true. Even if you didn’t know that as a fact, you certainly see that it is quite possible. The first open sentence is said to be functional, meaning that as long as you fill in the blank with designators for the same thing, all the different sentences you get will have the same truth value: Same thing, same truth value. The second open sentence is not functional, meaning that there are (at least) two designators that designate the very same thing, but that filling in the blank with one designator produces a true statement in some situation and filling in the blank with the other designator produces a false sentence in the same situation.

A predicate is a labeled, functional, open sentence.

### II. Statement and Predicate Forms

Simple statements that have no structure as far as SL is concerned usually have structure (designator/predicate structure) from the standpoint of PL. We introduce symbols to help use describe that structure by means of PL statement forms.

The lower case letters a,b,c, ..., t are called designator letters and are used to stand for designators. With these letters we can indicate the sameness and difference of the designators that occur in statements. We will use the same letter to indicate that the very same thing is being referred to by the designators involved, even though the words used may differ. We will use different letters to indicate that different things are being referred to, even though the words used may be the same.

The upper case letters A, ..., Z are predicate letters and are used to represent sameness or difference of predicates. We will often indicate by means of a symbol key which letters are used for which predicates and designators. For example, we might use the following symbols according to this key:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>Bob</td>
</tr>
<tr>
<td>c</td>
<td>Carol</td>
</tr>
<tr>
<td>P</td>
<td>is present</td>
</tr>
<tr>
<td>F</td>
<td>1 and 2 are friends of each other</td>
</tr>
</tbody>
</table>

©Harold Levin  Page 71  10/12/98
to describe these sentences:

Bob is present  
Bob is present, but Carol isn’t  
If Bob is present then so is Carol  
Bob and Carol are friends  
Bob and Carol are not friends

with these PL statement forms:

\[ P_b \]
\[ P_b \land \neg P_c \]
\[ P_b \rightarrow P_c \]
\[ F_{bc} \]
\[ \neg F_{bc} \]

Notice that we always symbolize the predicate and then follow the predicate letter with the number of designator symbols needed to fill in all the blanks in the predicate that the letter stands for. We continue to use truth function symbols as before. Now we can describe more of the details, but we include all of the old familiar truth functional structure information too.

Predicate Forms

Predicates are (as the name of this theory suggests) the major focus of our attention in PL. So we will be interested not only in symbolic descriptions of statements, but also in symbolic descriptions of predicates. We will encounter not only logical operators that operate on statement components to form compound statements, but also logical operators that operate on component predicates to form compound predicates. So we will introduce predicate forms to describe predicates.

An open statement form is just a statement form with designators removed and replaced by blanks. A predicate form is just a labeled open statement form. Here are some sample open statement forms:

\[ P_\_ \]
\[ P_\_ \land P_\_ \]
\[ P_b \rightarrow P_\_ \]
\[ F_\_ \_ \_ \]

Here are some sample predicate forms:

\[ P[1] \]
\[ P_b \rightarrow P[1] \]
\[ F[1][1] \]

Other Styles of Labels

It is not common to use boxed numerals as labels in predicates. There are other styles of label that are commonly used instead. One style uses the letter ‘x’ instead of a box and puts the numeral in a subscript position instead of inside the box. For example, each pair is just two different styles of writing the very same predicate:
1 is present
x₁ is present

1 is present, but 2 is not present
x₁ is present, but x₂ is not present.

Another style of labelling is to use letters such as u,v,w,x,y,z with or without numeral subscripts and to define an ordering on the letters (alphabetical, for instance) so that we can always talk about the first label, the second label, etc. in a particular predicate. For example,

y is present
z is present, but u is not present.

In the first case, there is only one label, so it corresponds to 1. In the second case, u is alphabetically earlier than z, so u corresponds to 1 and z to 2.

These last two styles of labelling are common. The letters x,y,z, etc. used are called variables. The correct, but not usually recognized, explanation of what a variable is: A variable is a device used to label the blanks in predicates.

III. Logical Analysis

Just as happened in SL, the logical operators of PL combine components with values to form compounds with values that can be computed once we know the value of the components and understand how the logical operators work.

Predicate Values

Predicates have values, but the values of predicates are not (except in special cases) truth values. Predicates are not true nor are they false. But predicates are true–of and false–of things. For example, the predicate,

1 is an even number

is true of 6 and false of 17. We will consider the collection of things that a predicate is true of to be the value of the predicate. Sometimes this value is called the “extension” of the predicate. Sometimes it is called “the relation expressed by the predicate.”

Things are not quite so simple as I have indicated so far. We have two small problems to face. First we have the problem of being more definite about what the candidate things are that might or might not be part of the value of a predicate. Second, we have to make special arrangements when the predicate has more than 1 place, when the predicate is a 2 or 3 (or more) place predicate.

Domains
Whenever we are interested in predicates and their values, we will always begin by picking out a set from which the things in the value of the predicate are to be found—we call this set the **domain**. For instance, we might choose (arbitrarily, and just for the sake of example) the set of numbers, 

\[ \{1, 2, 3, 4\} \]

as our domain. Then, with this domain chosen, the value of the predicate, 

1 is an even number

is this:

\[ \{2, 4\} \]

and the value of the predicate,

1 is an odd number

is this:

\[ \{1, 3\} \].

**Tuples and Relations**

When we consider 2-place predicates and their values, we have an extra problem. What are 2-place predicates true of? For instance, using the same domain as above, what is the value of,

1 is less than 2?

This 2-place predicate is not true of any single number. It is not true of the number 3, for example. But it isn’t false of the number 3 either. Rather, before we can talk of this predicate being true or false of something, we need two numbers, a pair of numbers, a 2-tuple of numbers. Still, even if we consider a pair of numbers chosen from our domain, for instance, 4 and 2, we need a way of linking up the numbers in the pair with the blanks in the predicate before we can say something true or false. Let’s consider the items of a pair to be listed in order from left to right—the first item on the left and the second item on the right. And let’s link up the first item of the pair with the first blank in the predicate and the second item of the pair with the second item of the predicate. So the pairs that our predicate is true of (chosen from the domain) are these:

1, 2
1, 3
1, 4
2, 3
2, 4
3, 4.

If we try to list the pairs all on the same line (or as few lines as possible) we have a small problem of telling the end of one pair and the beginning of another. To make this easier to tell at a glance, we
surround the items in a pair with the angle brackets as follows:

\(<1,2>, <1,3>, <1,4>, <2,3>, <2,4>, <3,4>.\)

Similar notation works for 3-tuples, 4-tuples, etc.

By the way, I didn’t exactly tell you the value of the predicates in the last examples, nor did we compute the values. We simply made use of our understanding of English and of our notation for predicates and their values to figure out the proper value and write it down.

A set of tuples such as we have decided to have as the value of predicates is called a relation. A relation consists of a collection of tuples all of the same number of elements—all 1-tuples, or all 2-tuples, etc. A relation is called 1-place if it consists entirely of 1-tuples, 2-place if it consists entirely of 2-tuples, etc.

The value of a 1-place predicate is a 1-place relation, the value of a 2-place predicate is a 2-place relation, etc.

**Values Of Complex Predicates**

Complex predicates have values that can be computed from the values of the component predicates. Consider for example (using the same domain as previously), the predicate,

\[\Box \text{ is odd} \land \neg 1 < \Box\]

which combines the two component predicates,

\[\Box \text{ is odd}\]

and

\[1 < \Box\]

using negation and conjunction. How do we compute (figure out) the value of the compound predicate? First, since the compound predicate contains two different labels, it is a 2-place predicate. So its value will be a 2-place relation, a collection of 2-tuples (pairs) of items from the domain. But what pairs? We figure this out by first considering all the possible pairs of items:

\(<1,1>, <1,2>, <1,3>, <1,4>, <2,1>, <2,2>, <2,3>, <2,4>, <3,1>, <3,2>, <3,3>, <3,4>, <4,1>, <4,2>, <4,3>, <4,4>.\)

Then we determine, for each of the possible pairs, whether or not the predicate is true of it. How do we tell whether the predicate is true of the pair \(<3,1>\), for example? We fill in the blanks correctly and determine whether we end up saying something true or false. That is, we determine whether

\[1 \text{ is odd} \land \neg 3 < 1\]
is true or false. It is true—because 1 is odd and 3 is not less than 1. And so the pair <3,1> does belong in the relation that is the value of the predicate.

In general: To compute the value of a logically complex predicate do three things in order. First, determine the “arity” of the predicate—how many places the predicate has. Though the labeling scheme used may vary from case to case, the arity is always the number of different labels used in the predicate. Second, list the complete set of all tuples of items from the domain with as many items in the tuple as the predicate has places. For a one–place predicate, list all one–tuples of items from the domain. For a two–place predicate, list all two–tuples of items from the domain. For a three–place predicate, list all three–tuples of items from the domain. And so on. Finally, for each tuple, determine whether it satisfies the predicate or not (whether the predicate is true of it or false of it). The tuples that satisfy the predicate belong in the value of the predicate; the ones that don’t satisfy the predicate don’t belong in the value of the predicate.

To tell whether a tuple satisfies a predicate, we form the “satisfaction statement” corresponding to the tuple and predicate. The satisfaction statement is formed by putting a designator for the first item in the tuple into every blank of the predicate labeled by label number 1, putting a designator for the second item in the tuple (if there is one) into every blank of the predicate labeled by label number 2, and so on until all the blanks in the predicate are filled. If the resulting statement is true, the tuple satisfies the predicate; if the statement is false, the tuple does not satisfy the predicate.

In particular cases, we may be able to find the value of a predicate with far less work than this general method would suggest.

IV. The Catalogue 2

Now we can complete the description of the catalogue of PL. The missing item is the new category of logical operator, quantifiers.

Quantifiers

Quantifiers are logical operators that operate on predicates. For the time being, we will consider just a special case, quantifiers applying to 1-place predicates.

A quantifier is a logical operator that operates on a 1-place predicate and makes a claim about the number of items in the domain that the predicate is true of. For example, a quantifier could
claim any of the following about a predicate:

- true of everything in the domain
- true of nothing in the domain
- true of some, but not all in the domain
- true of at least one item in the domain
- true of exactly five things in the domain.

In PL we concentrate on just two quantifiers, the universal quantifier and the existential quantifier.

The universal quantifier claims that the predicate to which it applies is true of everything in the domain. The existential quantifier claims that the predicate to which it applies is true of at least one thing in the domain.

### V. Statement Forms 2

We will use two special symbols to stand for the two quantifiers of PL:

- $\forall$ universal quantifier symbol
- $\exists$ existential quantifier symbol.

There is a special way in which these symbols are used. To apply, for example, an existential quantifier to a 1-place predicate (for example, the predicate, $x$ is even) we do three things:

- Write down the quantifier symbol.
- Write down the label used to mark the blanks in the predicate.
- Write down the predicate.

We will get these results respectively for the three steps in our example:

$$\exists x \ x \text{ is even.}$$

A similar series of steps leads to the universal quantification of the predicate, $y$ is odd:

$$\forall y \ y \text{ is odd.}$$

The first example claims that there is at least one even number in the domain. The second example claims that all the numbers in the domain are odd.

Quantifiers can combine with truth functions to form complex statements. This can happen in two ways. First, we can take any statement and negate it, or combine it in conjunction, disjunction, conditional, or biconditional with other statements. For example,

$$\neg \exists x \ x \text{ is even}$$

$$(\neg \exists x \ x \text{ is even} \land \forall y \ y \text{ is odd}).$$

Here the only new feature is that the component statements are logically complex and have a quantifier applied to a predicate.

The second sort of combination of truth functions and quantifiers occurs when the truth functions are used to express a logically complex predicate that the quantifier applies to. For example,
$\exists x \neg x$ is even \\
$\forall y (y$ is even $\lor y$ is odd).

Of course, we can have both kinds of combination at once:

$\neg \exists x \neg x$ is odd.

It is still important to be able to tell which logical operator is the main logical operator, and parentheses are still the means for indicating this information. In the following expressions, the main logical operator is boxed.

$\neg\forall x \ x$ is odd \\
$\neg \exists y \ y$ is even $\lor 7$ is odd \\
$\exists x \ (x$ is odd $\rightarrow \neg x$ is even) \\
$\exists y \ y$ is even $\rightarrow \exists z \ z$ is odd

VI. Logical Analysis 2

Now we can determine not only the value of predicates, simple and complex, but also the truth value of complex statements that involve all the logical operators, truth functions and quantifiers.

The basic idea is to follow these simple rules:

- Find the main logical operator.
- If the mlo is a truth function, determine the truth value of the components and combine in the correct way.
- If the mlo is a quantifier, determine the value of the predicate to which it applies and then see if the claim it makes is correct.

These rules may need to be used several times in the course of the computation, but they will eventually produce the correct result. For example, we will find the truth value of the statement,

$\neg \exists x \neg x$ is odd.

For these examples, the domain is $\{1, 2, 3, 4\}$.

First step: find the main logical operator. We have boxed it below.

$\exists x \neg x$ is odd

Since the mlo is a truth function, we will now concentrate on finding the truth value of the component. In this example, that means finding the truth value of,

$(1) \exists x \neg x$ is odd.

When we have done that, we know that the result we want is the negation of the truth value of this component.

To find the truth value of $(1)$ we apply our procedure all over again to it. Find
the main logical operator. We box it below.

$$\exists x \neg x \text{ is odd}$$

So, since we have a quantifier as mlo, we need to determine the value of the predicate to which it applies. That is, we need to know the value of,

$$\neg x \text{ is odd}.$$ 

In the case at hand, the value is \{2, 4\}. The quantified expression (1) says that the predicate is true of at least one thing in the domain, and we have just determined that this claim is correct. So (1) is true. But the answer we are looking for is the truth value of the negation of (1). In this case that is the negation of true, namely false. Our answer is, FALSE.

VII. Simple Symbolization

Our final topic for this initial look at PL is to see how English expresses the new logical operators, quantifiers, and to learn to represent the full PL structure of English sentences in the notation of PL. We begin this topic now and return to it later on.

English Quantifiers

Operator-Style

English has quantifier-expressing phrases that have a use very similar to our use of the symbolic quantifiers. For example, in English we can express

$$\exists x x \text{ is even}$$

in this way,

Some thing in the domain is such that it is even.

Here there are parts that correspond directly to the symbolic form and other parts that have no counterpart in symbols. The correspondence can be displayed in a table like this:

| $\exists$ | Some        |
| x        | thing in the domain |
| is such that |
| x        | it           |
| is even  | is even      |

Notice that even though English does not have special labels for the blanks in a predicate, the pronoun ‘it’ functions in this example as a variable, labeling the blank in the predicate.

Even though this logical operator style of quantifier is awkward sounding, it is to be preferred when our aim is symbolization (though not if we are after the Pulitzer Prize). Logicians, mathematicians, and philosophers talk in this odd way because it makes the job of describing the logical form of what is said easy, and thus aids in analyzing the logical
properties of what is said in that form. I will encourage you to also learn to speak this way.

There are logical operator style phrases for other quantifiers in English in addition to the existential quantifier. For example,

No thing is such that it is even,
Every thing is such that it is even.
The first of these is not one of PL’s two special quantifiers. The second is a case of universal quantification. However, we can express both in PL logical operator notation. The statement forms are, respectively,

\[ \neg \exists x \ x \ \text{is even}, \]
\[ \forall x \ x \ \text{is even}. \]

**Noun Phrase-Style**

It is more common in English to use noun phrase forms of quantifiers which fill in the blanks of the predicates to which they apply. For example,

Some thing is even,
Every thing is even,
No thing is even.

It is best to reword even such simple statements into the logical operator form before symbolizing. Let us consider a slightly more complex example to be symbolized:

Every thing is either even or odd.
First step, reword as logical operator quantification:

Every thing is such that either it is even or it is odd.

By the way, in the original, we can tell that it is the universal quantifier that is the mlo and not the disjunction. What grouping device helps us do that? Now symbolization is easy.

\[ \forall z (z \ \text{is even} \lor z \ \text{is odd}). \]

Why use ‘z’? Just to be different. We can use any variable. Always using ‘x’ is boring for me. Notice the complex predicate.

One more example:

Either every thing is even or every thing is odd.
In this case, the mlo is disjunction. So the first step is to get that much symbolized.

\[ (\text{every thing is even} \lor \text{every thing is odd}). \]

Now we can symbolize the components and stick the result back into the disjunction. The first component,

\[ \text{every thing is even} \]

is reworded,
every thing is such that it is even
and symbolized,

\( \forall x \ x \text{ is even.} \)

So too the second component,

everything is odd
is reworded,

everything is such that it is odd
and symbolized,

\( \forall x \ x \text{ is odd.} \)

Finally the two symbolizations of the
two components are placed into the
disjunction,

\( (\forall x \ x \text{ is even} \lor \forall x \ x \text{ is odd}). \)

Restricted Quantifiers

English has a large number of ways of
hiding quantifier/predicate/truth function
structure. But there are a few basic
patterns that account for the vast
majority of the cases we will ever
encounter. Most important of these are
the idioms of restricted quantification.

Sometimes we don’t want to involve the
entire domain in our quantificational
claim. I might want to claim that a
certain predicate was true of all of the
even numbers in my domain, but not to
claim that it was true of all the numbers
in that domain. For example, I might
want to claim that

\( x \text{ is bigger than 7} \)

is true of all the even numbers in the do-
main, but not of all the numbers in the
domain.

Indeed we often want to restrict our
quantifications in just this way. So we
want some way to accomplish this or to
describe it in our notation of PL
statement forms. There are many ways
that we might think of to accomplish the
aim of symbolically describing restricted
quantification. The one usually chosen,
is to discover a combination of truth
functions and quantifiers that expresses
the same thing that we express in
English with restricted quantification. It
is, perhaps, surprising that it can be
done, but it can be done.

We can express the restricted universal
quantification of the example above in
this fashion:

\( \forall x (x \text{ is even} \rightarrow \\
\quad x \text{ is bigger than 7}). \)

This “says” that every even number in
the domain is bigger than 7. The
symbolization does not describe the
actual words that were used to make the
statement, but instead describes the way
in which the value of the predicates
involved (‘x is even’ and ‘x is bigger
than 7’) combine to determine the truth value of the statement.

There are just a few common patterns of restricted quantification that occur over and over again in many different guises in English.

- All As are Bs,
- Some As are Bs,
- No As are Bs,
- Only As are Bs,
- Exactly the same things are As as are Bs.

If we understand these idioms and how to recognize variations in wording that express the same ideas, and how to represent in the notation of PL these restricted quantifications, we will understand about 80% of PL symbolization.

Here are the common PL logical operator combinations used to express these basic restricted quantifications:

\[
\begin{align*}
\forall x (x \text{ is } A \rightarrow x \text{ is } B), \\
\exists x (x \text{ is } A \land x \text{ is } B), \\
\neg \exists x (x \text{ is } A \land x \text{ is } B), \\
\forall x (x \text{ is } B \rightarrow x \text{ is } A), \\
\forall x (x \text{ is } A \leftrightarrow x \text{ is } B).
\end{align*}
\]

Now when we symbolize, we will have a longer list of mlo types to look for: negation, conjunction, disjunction, conditional, biconditional, neither…nor, universal quantification, existential quantification, and the restricted quantifications. After determining the “mlo” form, we will symbolize the logical operator structure leaving the components expressed in English. Then we will concentrate on each component in turn, symbolizing it, and placing its symbolization correctly within the overall logical operator structure.