6.4. Describing models

6.4.0. Overview

The grammatical variety we have been considering brings with it a greater variety of semantic values; in particular, the counterexamples to arguments that are not formally valid are now more than simple assignments of truth values.

6.4.1. Extensions and ranges
While the extensions of individual terms are (like the extensions of sentences) single values, the extensions of operators are (like the extensions of connectives) functions.

6.4.2. Building structures
The extensions of predicates (functions from reference values to truth values) can be fully specified by telling the input for which they give output T, and these cases can be presented in diagrams to which the extensions of other items of non-logical vocabulary can be added.

6.4.3. Structures as counterexamples
Although extensional interpretations are now different, it is still true that a dead-end open gap specifies the sort of counterexample that lurks in the gap.

Glen Helman 01 Aug 2013

6.4.1. Extensions and ranges

In this section, we will look at ways of describing the semantic values of the new sorts of expression we have been considering and ways of using these values to present counterexamples to derivations that fail. First, let us collect and sharpen what we know about the semantic values of the several kinds of expression we are considering. Table 6.4.1-1 gives a basic summary that you may compare with the tables of grammatical categories given in 6.1.1 and 6.1.7.

Expression	*Extension*		
sentence	truth value		
term	reference value		
		input	*output*
connective	truth function	truth value(s)	truth value
predicate	property or relation	reference value(s)	truth value
functor	reference function	reference value(s)	reference value

Table 6.4.1-1. The extensions of 5 kinds of expression.

In each case the *intension* of an expression is a specification of its extension in each possible world. For example, the intension of an individual term is specifies its reference value in each possible world; this is the sort of intensional entity that was mentioned in 6.3.1. In particular, while Barack Obama and the U. S. president have the same extension in the actual world, they have different intensions because their extensions differ in other possible worlds.

Since the extensions of the incomplete expressions are functions, they exhibit generality: each such extension determines an output value for each input value from some range of such values. In the case of connectives, the input values are fixed as the two truth values **T** and **F**, and the range of generality of truth functions is thus quite limited. We do not fix the range of reference values, but this range must be known before we know what functions are available as extensions of predicates and functors. We will refer to a specification of the reference values as a *referential range* or often simply as a *range*, and we will use the symbol **R** for it. (The word domain is often used for this idea, but we will use that word for another concept.) The referential range can be any set that is not empty.

Our logical constants have fixed extensions that we stipulate once and for all. In the case of connectives these are given by their tables. The identity predicate = has an extension that is settled once the referential range is settled: this predicate is true of any pair of reference values whose members are the same but false of any pair of different values. Further basic expressions—unanalyzed sentences, unanalyzed terms other than variables, unanalyzed predicates, and unanalyzed functors—form our *non-logical vocabulary*, and their extensions are not fixed.

As in truth-functional logic, items of non-logical vocabulary may be assigned extensions by *extensional interpretations* or assigned extensions for all possible worlds by *intensional interpretations*. The extensions assigned to predicates and functors by a given interpretation must have a generality that extends to the same range **R**, so we will speak of an extensional interpretation as being an *interpretation on* a range **R**. The basic semantic information needed for the logical forms we are now considering is then a range **R** and an extensional interpretation (on that range) of certain items of non-logical vocabulary; we will refer to this information as a *structure for* any expressions that can be formed from this non-logical vocabulary and our logical vocabulary.

It will be convenient to assume that every reference value in the range of a structure comes with a label. We will refer to this label as the *ID* of the value. The assumption that all reference values have IDs is actually quite a heavy one. The limitations of decimal notation in capturing irrational numbers like π and the square root of 2 are essential; no system of finite expressions could name all real numbers. So if a range includes all real numbers, its members could not all be labeled by expressions in any ordinary sense. One way around this is to think of IDs as mathematical abstractions—for example, as ID numbers that are not merely numerical expressions but genuine numbers. In this way, the real numbers might be used as their own IDs. But, while these are important theoretical issues that have had considerable impact on the development of logic, they will not affect us practically. Our chief interest will be in structures indicated by the dead end gaps of derivations; and these will all have finite (and usually very small) ranges, so there will be no problem in using numerals (even single-digit numerals) as their IDs.

Glen Helman 01 Aug 2013

6.4.2. Building structures

Once a referential range **R** is specified, the extensions of the various sorts of non-logical vocabulary can be specified in ways that extend the approach used in truth-functional logic. Individual terms are merely assigned reference values from **R** in the way sentences were assigned truth values. The extensions of predicates and functors are functions and, as we have already seen, these can be indicated by tables analogous to truth tables. Below is an example of an extensional interpretation of the following non-logical vocabulary:

sentences: A, B
individual terms: a, b, c, d, e
predicates: F (*1-place*), G (*1-place*), R (*2-place*)
functors: f (*2-place*)

We choose (arbitrarily) a referential range with five values whose IDs run from 0 to 4 and assign (again arbitrarily) extensions of the appropriate sorts to the items of non-logical vocabulary.

R: 0, 1, 2, 3, 4

A	B
T	F

a	b	c	d	e
3	4	0	2	4

τ	Fτ
0	F
1	T
2	T
3	F
4	T

τ	Gτ
0	T
1	F
2	F
3	T
4	T

R	0	1	2	3	4
0	F	F	F	F	F
1	T	F	T	F	T
2	F	T	F	F	T
3	F	T	F	T	F
4	F	F	F	F	T

f	0	1	2	3	4
0	1	2	3	1	4
1	2	4	0	1	3
2	3	1	2	0	1
3	4	1	0	3	0
4	4	3	1	2	4

The truth-table row giving the values of A and B is dwarfed by the other information in the structure, but the whole of the structure has the same significance for our present analysis of logical form as did the assignment of truth values in truth-functional logic.

Since we build in no assumptions about the size of the range **R**, we can consider structures that are quite small, and it is possible to represent small structures in pictorial diagrams. As in Figure 6.4.2-1, let us depict a range by a rectangle, with the values of the range shown as circles that enclose numbers, which will serve as the IDs of the reference values in the range.

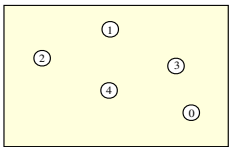


Fig. 6.4.2-1. A range of reference values labeled by their IDs.

The extension of a simple term will be one of these reference values, and we can show this by writing the term next to that value. Figure 6.4.2-2 shows the extensions assigned above to the terms a, b, c, d, and e. The terms b and e are

written next to the same value because both were assigned that value as their extension.

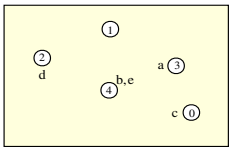


Fig. 6.4.2-2. A range with the extensions of five terms shown; two have the same extension.

The extension of a one-place predicate is a function that yields a truth value as output when it is applied to a reference value as input. We will say that it is *true* or *false of* a reference value depending on its output for that value as input. We can represent this sort of extension by writing the predicate next to the values it is true of; we will then know that it is false of any other reference values. Figure 6.4.2-3A does this for the predicates F and G, again using the extensions originally given in tables.

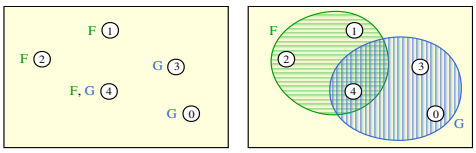


Fig. 6.4.2-3. A range with the extensions of two one-place predicates indicated by labeling values (A) and enclosing sets of values (B).

This can make the diagram rather cluttered, but we can clean things up a little by drawing a line around all the values a predicate is true of and labeling the line rather than the values. This is done in Figure 6.4.2-3B. The second sort of diagram corresponds fairly directly to what was the original concept of an extension—the class of things a predicate is true of—and we can say that the values a predicate is true of are *in* its extension.

Predicates with more than one place are not true or false of single values but of pairs, triples, or longer series of values. For example, [_ is the father of _] is true not of James Mill or of John Stuart Mill (his son) taken individually but instead of the two taken together and in that order. Such an *ordered pair* of values can be represented in our diagrams by an arrow from its first to its second member. (We could represent longer series of values by adding legs between the head and tail of the arrow.) The extension of a 2-place predicate can be thought of as the collection of pairs it is true of. Similarly, the extension of a 3-place predicate will be a collection of triples, and the extension of a predicate with some number *n* of places will be a collection of *n-tuples* (i.e., of series with length *n*).

There are a number of ways a collection of *n*-tuples can be depicted. We might draw the arrows that represent its members and label each one as we initially labeled the values in the extension of a one-place predicate. This would make for some more clutter, but it would be hard to avoid that by drawing a line around a group of arrows. We might write the predicate once and draw a line from it to each of the arrows in its extension, or we might draw different styles of arrows for different predicates, labeling the style of arrows in a legend like that of a road map. Each of these three approaches to labeling the extensions of many-place predicates has its value but we will most often use legends. Figure 6.4.2-4 shows this latter style of diagram for the extension that was assigned to the 2-place predicate R.

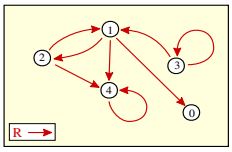


Fig. 6.4.2-4. A range with the extension of a 2-place predicate indicated.

Notice that the predicate is true of the values 1 and 2 taken in either direction and that it is also true of each of 3 and 4 paired with itself (and look back to see how that information appeared in the table).

Figure 6.4.2-5A combines the extensions of the three predicates, and Figure 6.4.2-5B adds the interpretation of the unanalyzed sentence A.

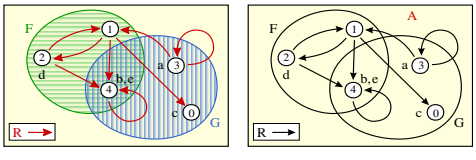


Fig. 6.4.2-5. A range with the extensions of predicates (A) and predicates together with the indication that a sentence is true (B).

As was noted in 6.2.2, unanalyzed sentences can be thought of as zero-place predicates. That means that they do not express properties that may or may not be true of objects or relations that may or may not hold between objects. Instead sentence express state of affairs that are simply true or false. One way to indicate that in a diagram is to simply include a true sentence within the rectangle, understanding any unanalyzed sentence that does not appear there to be assigned the value **F**.

All that is left are the extensions of functors. We could use arrows here, too, because a reference function establishes a relation between its input and output values. For example, the squaring function relates 1 to itself, 2 to 4, 3 to 9, and so on. However, this would make for a lot of arrows. A one-place function relates each value to some other value (perhaps the Nil), so each value would be at the tail of an arrow; and things get much worse with functions of two or more places. We can get a somewhat more compact notation by adapting the way we indicate the extensions of individual terms. Next to each output value of a functor, we can write the functor with its places filled by IDs of the input values for which it yields that output (for example, writing f01 next to the value 2 to say that 2 is the output of f for inputs 0 and 1). Since the extension of a functor may yield the same output for different input values, we may need to write the functor next to an ID several times, each time filling its places with the IDs of different input values. This is manageable for 1-place functors because, for each such functor, we will need only as many labels as there are possible input values—i.e., one for each member of the range. But for a function with two places, the number of labels is the square of the number of reference values and this number mounts rapidly. In the example we are considering, we would need to write the 2-place functor f 25 times to indicate all 25 entries of the table shown earlier. Adding to these only the individual terms leads to the rather cluttered diagram shown in Figure 6.4.2-6.

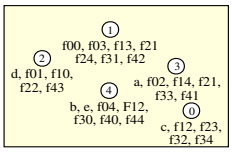
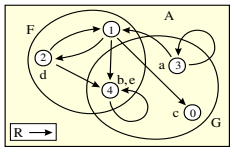


Fig. 6.4.2-6. A range showing the extensions of a 2-place functor and several individual terms.

Most of the structures we consider will be quite small, and this approach will be more feasible with them. Still, it is always possible to supplement a diagram with one or more tables, and that is the easiest approach for the example we have been considering. The full interpretation is given in this way in Figure 6.4.2-7.



f	0	1	2	3	4
0	1	2	3	1	4
1	2	4	0	1	3
2	3	1	2	0	1
3	4	1	0	3	0
4	4	3	1	2	4

Fig. 6.4.2-7. A structure for a variety of non-logical vocabulary.

Now let us do something with this structure. Interpretations are assigned in order to settle the truth values of sentences formed using the vocabulary that is interpreted. This is analogous to calculating a truth value of a truth-functional compound given an assignment of truth values to its ultimate components. Below is the calculation of the truth value given to a sentence by the structure we have been considering. It is followed by an explanation of the initial steps in the process; this explanation refers to the way the interpretation is presented in the diagram and table in Figure 6.4.2-7.

(G a ∧ R d e) → ((F (f a b) ∨ ¬ B) ∧ b = e)
T 3 T T 2 4 Ⓣ F 0 3 4 T T F T 4 T 4

Ga: a has 3 as its value and this value is in the area representing the extension of G, so Ga gets **T**

Rde: d and e have 2 and 4 as their extensions and the arrow for this pair is in the extension assigned to R, so Rde gets **T**

F(fab): f yields the value 0 when given the extensions of a and b (the values 3 and 4) as input (as can be seen from the end of the next-to-last row of the table for F), and 0 is not in the area marked as the extension of F; thus fab gets 0 and F(fab) gets **F**

B: B gets the value **F** since, unlike A, it does not appear within the rectangle

b = e: b and e both have 4 as their extension, so b = e gets **T**

The extensions of complete unanalyzed expressions have been written under these expressions, and the values of compounds are written under signs for the operators that form them. As in truth-functional logic, the order of calculation is determined by parentheses. Notice that capital letters always have truth values under them and lower case letters always have reference values under them.

Glen Helman 01 Aug 2013

6.4.3. Structures as counterexamples

Since structures provide the information that is now needed to determine truth values for sentences, we will present counterexamples to derivations that fail by describing structures. An example of a failed derivation is shown below.

	P(fa)b → Qa(fd) Qbd → Fb b = d	3 5 a, b—d, fa, fd
	P(fd)d ∧ a = d	2
2 Ext 2 Ext 3 MPP=	P(fd)d a = d Qa(fd)	(3) a—b—d, fa—fd
	¬ Fd	(5)
5 MTT=	¬ Qbd ○	b=d,P(fd)d,a=d,Qa(fd),¬ Fd,¬ Qbd ⊭ ⊥
	⊥	4
4 IP	Fd	1
1 CP	(P(fb)d ∧ a = d) → Fd	

Stage 3 of the development uses the extended version of *modus ponens*. At this point, we have two alias sets, one consisting of a, b, and d and the other consisting of fa and fd. We do not have the antecedent of the conditional P(fa)b → Qa(fb) among our resources but rather a sentence, P(fd)d, that, although differing from it in two places, differs only by terms that are co-aliases for fa and b. Stage 5 uses a similarly extended *modus tollens*. The remaining open gap cannot be closed because Qa(fd) and ¬ Qbd, the two resources that might be part of a contradiction, differ in their second place by terms (fd and d) that have not been made co-aliases.

The active resources of the dead-end gap form the consistent set:

b = d, P(fd)d, a = d, Qa(fd), ¬ Fd, ¬ Qbd

To describe a structure making the members of this set true, we must choose a range of reference and assign an extension to each of the items of non-logical vocabulary. The choice of the referential range and the assignment of extensions to both individual terms and functors is determined by the alias sets. We choose one reference value for each alias set and assign extensions so that the terms in the set have that as their reference value.

For this consistent set, we will have two alias sets, one containing a, b, and d and the other containing fa and fd, so we take the range to consist of two values, one corresponding to each alias set. We do this by numbering the alias sets and taking these numbers to be the IDs of the values in the range.

Next we must assign values to non-logical vocabulary appearing in the terms in such a way that each term has the reference value corresponding to the number of its alias set. In the case of an unanalyzed term, we simply assign it the value of its alias set. In the case of a compound term, we place the following constraint on the reference function used to interpret its main functor (the one used last in forming it): the output of the reference function must be the reference value of the compound term when the input consists of the reference values of the component terms. That is to say, we need to insure that we will get to same output reference value from given input terms whether we take go across the top of the following diagram and then down the right side or go down the left side first and then across the bottom—or, as a mathematician would say, the diagram must "commute" (i.e., the order of across and down must be reversible):

	input		*output*
syntax	term(s)	→	compound term
	↓		↓
semantics	reference value(s)	→	reference value

The arrow across the top is the operation of applying a given functor while the arrows going down are settled by the numbering of the alias sets of term. Finding compound terms in alias sets then tells us something about how the reference function used to interpret the functor associates output with certain input.

In the example we are looking at, the two compound terms bring with them the same constraint of this sort since they are co-aliases and have components which are co-aliases. The table below shows the association of ID numbers with alias sets and the constraints on the interpretation of functors that follow from this association:

term	*ID*	*constraint*
a	1	a: 1
b		b: 1
d		d: 1
fa	2	f1: 2
fd		f1: 2

To indicate constraints, we use a variation on the notation that was used to indicate the extensions of functors in the diagrammatic presentation of structures. Here "f1: 2" says that interpretation of f must yield output with ID 2 for input with ID 1.

In the case of predicates, the diagram that must commute is the following:

	input		*output*
syntax	term(s)	→	predication
	↓		↓
semantics	reference value(s)	→	truth value

Here the arrow at the top is the application of a given predicate, and the one at the bottom corresponds to the way the predicate's extension divides reference values into those the predicate is true of and those it is false of. The arrow down at the left is again given by the numbering of alias sets, and the one at the right is given by the presence of a predication or its denial in the set of sentences we are trying to make all true. What we know of the arrows going down then places constraints on the arrow across the bottom for certain predicates.

In the example, we have three non-logical predicates to consider, the 2-place predicates P and Q and the 1-place predicate F. Each sentence in the consistent set that affirms or denies one of these of a series of terms provides a constraint on the interpretation of that predicate—as is shown in the following table.

resource	*constraint*
P(fd)d	P21: **T**
Qa(fd)	Q12: **T**
¬ Qbd	Q11: **F**
¬ Fd	F1: **F**

The sentence P(fd)d tells us that P is true of values 2 and 1 (in that order) since these are the values of fd and d, respectively; but no other sentence says anything about the extension of P. There are sentences that require that the predicate Q be true of the pair 1 and 2 and false of the pair 1 and 1, but nothing is said about other cases. The last sentence requires that F be false of 1 but requires nothing beyond this.

The tables below incorporate this information about extensions. The values in grey are not required to make the members of the consistent set true and may be assigned arbitrarily. In the case of predicates, the value **F** has been assigned in such cases to make the extension as small as possible.

R: 1, 2

a	b	d
1	1	1

τ	fτ
1	2
2	1

τ	Fτ
1	F
2	F

P	1	2
1	F	F
2	T	F

Q	1	2
1	F	T
2	F	F

The upshot of these tables is depicted in Figure 6.3.4-1.

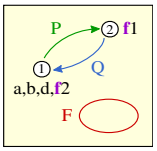


Fig. 6.4.3-1. A counterexample lurking in the open gap of the derivation above.

Since the predicates P and Q are each true of only one pair, they are used to label arrows directly. The emptiness of F's extension is shown by using F to label a circle that encloses nothing. This structure is small enough that the extension of the functor f is also represented in the diagram.

Much of the work here comes in assigning interpretations to individual terms and functors on the basis of a collection of alias sets. Let us look at another example of that. The example we worked out in 6.3.2 would arise if we were to check the entailment

a = b, fb = c, fb = fc, d = gca, g(fa)b = e ⊨ a = fd

The derivation for this is not very interesting. A single use of IP would leave us with a dead-end open gap which fails to close because

a = b, fb = c, fb = fc, d = gca, g(fa)b = e, ¬ a = fd ⊭ ⊥

The alias sets we found in 6.3.2 are shown below along with the corresponding constraints on the interpretation of individual terms and functors:

term	*ID*	*constraint*
a	1	a: 1
b		b: 1
c	2	c: 2
fa		f1: 2
fb		f1: 2
fc		f2: 2
fd	3	f4: 3
d	4	d: 4
e		e: 4
gca		g21: 4
g(fa)b		g21: 4

As in the example above, an unanalyzed term is simply assigned the number of its alias set. For a compound term, we require that the number of the alias set be the output value corresponding to input(s) that are the numbers of the alias sets of its immediate components. For example, the term fa appears in set 2, so

we want the table for f to lead us to calculate 2 as the reference value of fa. The input for the calculation will be the reference value of the term a; but a appears in set 1, so we want the table for f to yield output 2 for input 1. We derive exactly the same information from the appearance of the term fb; the output is the same because it appears in the same alias set as fa, and the input is the same because the term b appears in the same alias set as the term a. On the other hand, the appearance of fc in alias set 2, tells us that the table for f should assign output 2 also for input 2 since 2 is the alias set of the term c. We respond to the remaining terms in a similar way, the only difference being the need to note pairs of input values in the case of the 2-place functor g.

When we put constraints in tables assigning extensions to the individual terms a, b, c, d, and e the functors f and g, we get the following:

R: 1, 2, 3, 4

a	b	c	d	e
1	1	2	4	4

τ	fτ
1	2
2	2
3	
4	3

g	1	2	3	4
1				
2	4			
3				
4				

Many entries are left unfilled because they did not correspond to any terms in our alias sets. But, by the same token, we will never use these entries to calculate the values of terms appearing in the open gap, so they can be filled in arbitrarily. The value 1 is used in the tables below but any other would do; it is the other values that are significant.

R: 1, 2, 3, 4

a	b	c	d	e
1	1	2	4	4

τ	fτ
1	2
2	2
3	1
4	3

g	1	2	3	4
1	1	1	1	1
2	4	1	1	1
3	1	1	1	1
4	1	1	1	1

Recall that, in a couple of cases, we have had a single input-output pair dictated by two different terms. This raises the question whether the procedure we are using could ever lead to impose incompatible requirements? That is, could we end up trying to associate two different output values of a functor with the same series of input values and thus to fill in one entry in two different ways? For this to happen, there would have to be terms $f\tau_1 \ldots \tau_n$ and $f\upsilon_1 \ldots \upsilon_n$ with a common functor f that fell into different alias sets (if we were to have two output values), and the corresponding components of these compounds (τ_i and υ_i for i from 1 to n) would have to fall in the same alias sets (if we were to have the same input values in the two cases). But the way we have set up alias sets insures that this cannot happen. Instruction (iv) for drawing links would have told us to put the two compounds in the same alias set once their corresponding components were connected. And, indeed, in the two cases where we have duplicate requirements, the compounds appear in the same alias set precisely because we followed this instruction when forming the alias sets of this example. (Although terms whose corresponding components are co-aliases are bound to appear in the same alias set, they might do so for other reasons, too; for example, we might have both a = b and fa = fb as resources of a dead-end gap.)

We have now done enough to settle the truth values of all equations that appear affirmed or negated among the premises we are trying to make true. Do these values come out as we would like? That is, do the affirmed equations come out true and the negated ones false? Well, since the extensions given to all terms, simple or compound, will correspond to their alias sets, we know that any equation $\tau = \upsilon$ that is affirmed among the premises will be true. For such an equation will have led us to put the terms τ and υ into the same alias set, and each term will be assigned the value corresponding to this set as its extension. And, since they have the same extension, the equation between them will be true. How about the denial of an equation, a resource of the form $\neg\, \tau = \upsilon$? Since the gap cannot be closed, we know that τ and υ are members of different alias sets. And since the extensions given to these terms correspond to their alias sets, they will have different reference values and the equation $\tau = \upsilon$ will be false, making the resource $\neg\, \tau = \upsilon$ true—as is the case with ¬ a = fd in the example above.

We have been focusing on functors and equations since that is all that matters for the example, but similar considerations apply to non-logical predicates and predications of them. In the case of such predicates, it is our rules for closing gaps insure that we can assign interpretations consistently. If the gap cannot be closed we know that it does not contain both $P\tau_1 \ldots \tau_n$ and any sentence $\neg\, P\upsilon_1 \ldots \upsilon_n$ where the corresponding terms are co-aliases. And this means it never contains both an affirmation and a denial of P of any series of terms whose corresponding members are in the same alias sets. This means that we will never be led to require the extension of P to yield two different outputs for the same input. And the requirements we place on the extensions of non-logical predicates are designed to insure directly the truth of sentences affirming or denying the predication of such a predicate, so it is enough to know that our requirements are consistent to be sure that they will have the desired result.

The procedure we have been following enables us to find a counterexample lurking in any dead-end open gap, and the safety of our rules tells us that the same structure will separate the initial premises the derivation from its initial conclusion. We will generally not go on to confirm that it does since the calculations can be tedious, so the final step in showing that an entailment fails will be merely to *present a counterexample*—that is, to describe it using a diagram or tables.

Now the existence of a structure separating the premises from the conclusion is the test of formal validity of an argument. That is, if there is a structure that separates an argument's premises from its conclusion, then there is an intensional interpretation of it producing an actual English argument and a possible world that will separate the premises from the conclusion of that argument. This was easy to see in truth-functional logic, but more needs to be said in the case of the more complex interpretations we are now considering.

We cannot, as in 2.3.1, simply choose the actual world as the possible world that separates premises from conclusion because a structure, such as the one in Figure 6.4.3-1, may have only a limited number of reference values, while the actual world has many things in it (infinitely many if numbers are counted). The easiest approach in the present setting (but one that will no longer work in the next chapter) is to note that our calculations of extensions for the terms we are interested in remain the same in the presence of further reference values. When we chose a referential range, we could have added reference values that did not correspond to alias sets. Such values would not have played a role in the constraints on the interpretation of non-logical vocabulary or in the calculations of the values of components of the premises and conclusion of the argument we are interested in. So they would have neither contributed to nor interfered with the task of separating the premises from the conclusion. The possibility of adding such further reference values means that we can regard a structure like that of Figure 6.4.3-1 as a depiction of the way things stand for certain reference values *among others*. Given this understanding of a structure, it is not too hard to concoct intensional interpretations of the non-logical vocabulary that have the right extensions in the actual world. We might, for example, choose language describing an illustration of the structure. To capture the structure of 6.4.3.1, the interpretation of the term a could be the point labeled a and the interpretation of P could be [a P-arrow runs from _ to _]. If we use this sort of interpretation, drawing the structure is a way of making the actual world separate the argument's premises from its conclusion.

Glen Helman 01 Aug 2013

6.4.s. Summary

1 Logical forms (without free variables) may be given semantic values by assigning values to the non-logical vocabulary they contain; that is, they can be given extensions (or intensions) by an extensional (or intensional) *interpretation* of this vocabulary. The extensions of predicates and functors are functions that take as input reference values from a referential range **R** that must be specified along with an extensional interpretation; the range and the interpretations of non-logical vocabulary together constitute a structure for any expressions formed using only the non-logical vocabulary that is interpreted in the structure. We assume each value of the range is labeled by an ID.

2 The extensions of non-logical vocabulary can be represented using tables. In a more graphic approach, a referential range may be depicted by points in a plane labeled by their IDs, and further labeling and other devices can depict extensions of non-logical vocabulary on this range. For example, one-place predicates may label the points they are true of either individually or by labeling a line enclosing them. This set of points is one way of representing the extension of the predicate. If a predicate has more than one place, its extension must be a set of ordered pairs, triples, or other *n-tuples*; these may be represented by arrows (perhaps with legs) that indicate the order of values in the *n*-tuple. We may calculate the extensions that structures give to expressions by using a table analogous to a truth table, with all the information in a structure providing the basis for the calculation of a single row.

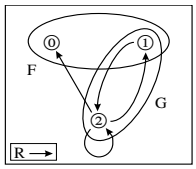
3 Structures are now the appropriate counterexamples to claims of validity. To build a structure that constitutes a counterexample lurking in a dead-end gap, we take the alias sets of the gap and choose a range that contains a value corresponding to each alias set. Then we assign extensions to unanalyzed terms and functors so that the reference value each compound term will be the value corresponding to the term's alias set. Finally, we assign extensions to predicates by seeing what terms the resources affirm or deny these predicates of. Our new rules for closing gaps ensure that these instructions are consistent and that a structure built in this way will lurk in the dead-end gap. The safety of the rules for developing gaps insures that it also lurks in the initial gap; but we will not go on to confirm this, so presenting a counterexample by describing such a structure will be the final step in showing that an entailment fails. Such a structure will also form at least a part of some possible world.

Glen Helman 01 Aug 2013

6.4.x. Exercise questions

1. Each of **a**, **b**, and **c** gives a structure in one of the two sorts of presentation described in this section—by a diagram or by tables. Present each of them in the other way.

a.



b.

τ	Fτ
0	T
1	T
2	F

τ	Gτ
0	F
1	F
2	T

R	0	1	2
0	T	T	T
1	F	T	F
2	F	T	T

c.

τ	Fτ
0	T
1	T
2	F

τ	Gτ
0	F
1	T
2	T

τ	Hτ
0	T
1	F
2	T

R	0	1	2
0	F	T	F
1	T	F	F
2	F	T	F

2. Calculate a truth value for each of the following sentences on the structure used as the chief example in this section (see, for example, Figure 6.4.2-7):

a. (Fa ∨ Gb) → Rab

b. R(fca)(fac)

c. fab = fba

3. Use derivations to check each of the claims below; if a claim of entailment fails, use either tables or a diagram to present a counterexample that lurks in an open gap.

a. a = a → Fa ⊨ Fa

b. ¬ (Fa ∧ Fb) ⊨ ¬ Fa → ¬ Fb

c. a = b ∨ b = a ⊨ a = b ∧ b = a

d. Fa → a = b, ga = b, Ra(ga) → Fa, F(ga) ⊨ Raa → R(ga)(ga)

e. a = b → Rac, ¬ a = b → Rbc ⊨ Rbc

For more exercises, use the exercise machine.

Glen Helman 01 Aug 2013

6.4.xa. Exercise answers

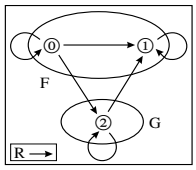
1. **a.**

τ	Fτ
0	T
1	T
2	F

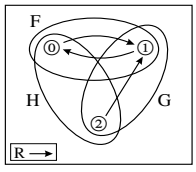
τ	Gτ
0	F
1	T
2	T

R	0	1	2
0	F	F	F
1	F	F	T
2	T	T	T

b.



c.



2. **a.**

(F	a	∨	G	b)	→	R	a	b
F	3	T	T	4	Ⓕ	F	3	4

b.

R	(f	c	a)	(f	a	c)
Ⓣ	1	0	3	4	3	0

c.

f	a	b	=	f	b	a
0	3	4	Ⓕ	2	4	3

3. **a.** *Without attachment rules:*

	a = a → Fa	2
	¬ Fa	(2)
2 MTT	¬ a = a	(3)
	●	
3 DC	⊥	1
1 IP	Fa	

Using attachment rules:

	a = a → Fa	2
1 CE	a = a	X,(2)
2 MPP	Fa	(3)
	●	
3 QED	Fa	

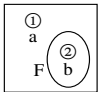
b.

	¬ (Fa ∧ Fb)	3
	¬ Fa	
	Fb	(3)
3 MPT	¬ Fa	
	○	¬ Fa,Fb ⊭ ⊥
	⊥	2
2 RAA	¬ Fb	1
1 CP	¬ Fa → ¬ Fb	

range: 1, 2

a	b
1	2

τ	Fτ
1	F
2	T



¬	(F	a	∧	F	b)	/	¬	F	a	→	¬	F	b
Ⓣ	F	1	F	T	2		T	F	1	Ⓕ	F	T	2

c.

	a = b ∨ b = a	1
	a = b	a—b
	●	
3 EC	a = b	2
	●	
4 EC	b = a	2
2 Cnj	a = b ∧ b = a	1
	b = a	a—b
	●	
6 EC	a = b	5
	●	
7 EC	b = a	5
5 Cnj	a = b ∧ b = a	1
1 PC	a = b ∧ b = a	

d.

	Fa → a = b	3
	ga = b	a, b—ga
	Ra(ga) → Fa	5
	F(ga)	
	Raa	(6)
	¬ R(ga)(ga)	(6)
	¬ Fa	(5)
5 MTT	¬ Ra(ga)	
	○	b=ga,F(ga),Raa,¬ R(ga)(ga), ¬ Fa,¬ Ra(ga) ⊭ ⊥
	⊥	4
4 IP	Fa	3
	a = b	a—b—ga
	●	
6 Nc=	⊥	3
3 RC	⊥	2
2 IP	R(ga)(ga)	1
1 CP	Raa → R(ga)(ga)	

range: 1, 2

a	b
1	2

τ	gτ
1	2
2	1

τ	Fτ
1	F
2	T

R	1	2
1	T	F
2	F	F

F	a → a = b , g	a = b , R	a (g	a) → F	a , F (g	a) / R	a	a → R (g	a) (g	a)
F	1 Ⓣ 1	F	2	2 1 Ⓣ	2	F	1	2	1 Ⓣ	F	1	Ⓣ	2	1	T	1	1 Ⓕ	F	2	1	2	1

e.

	a = b → Rac	3
	¬ a = b → Rbc	2
	¬ Rbc	(2),(4)
2 MTT	a = b	a—b, c; (3)
3 MPP	Rac	(4)
	●	
4 Nc=	⊥	1
1 IP	Rbc	

Glen Helman 01 Aug 2013