ENTAILMENT

THE LOGIC OF RELEVANCE AND NECESSITY

by

ALAN ROSS ANDERSON

and

NUEL D. BELNAP, JR.

and

J. MICHAEL DUNN

with contributions by

KIT FINE
ALASDAIR URQUHART

and further contributions by

DANIEL COHEN
STEVE GIAMBRONE
DOROTHY L. GROVER
ANIL GUPTA
GLEN HELMAN
ERROL P. MARTIN
MICHAEL A. MCROBBIE
STUART SHAPIRO

and including a Bibliography of Entailment by

ROBERT G. WOLF

VOLUME II

PRINCETON UNIVERSITY PRESS

Entailment

The Logic of Rel

VOLUME II

Alan Ross Ande
and J. Michael I

In spite of a pow
two thousand ye
ment the premis
conclusion, twer
glected the conc
publication of V
work. Since that
achieved an imp
philosophy: Volu
to a conclusion a
presentation of t
top people worki

Originally the
ply to cover cert
first volume—qu
or to extend the
such as semantic
technical progres
the publication o
II now includes
contains the wor
who has shown t
systems of releva
and of Kit Fine,
although the first
plete with respec
stant domain sem
plete with respec
"arbitrary object
important work b
authors, who are
Giambrone, Dor
Gupta, Glen Hel
Michael A. McR
Robert G. Wolf'
items is a valuab

Alan Ross An
Professor of Phil

(continued on the bac

Published by Princeton University Press, 41 William Street,
Princeton, New Jersey 08540
In the United Kingdom: Princeton University Press, Oxford

Library of Congress Cataloging-in-Publication Data

(Revised for vol. 2)

Anderson, Alan Ross.
Entailment: the logic of relevance and necessity.

Vol. 2. written by Alan Ross Anderson, Nuel D. Belnap and
J. Michael Dunn with contribution by Kit Fine . . . et al.
Includes bibliographical references.
1. Entailment (Logic). I. Belnap, Nuel D., 1930– —joint author.
II. Dunn, J. Michael, 1941– III. Fine, Kit. IV. Title.
BC199.E58A53 1975 160 72-14016
ISBN 0-691-07192-6 (v. 1) ISBN 0-691-07339-2 (v. 2)

This book has been composed in Monotype Times Roman by
Syntax International Pte Ltd

Princeton University Press books are printed on acid-free paper, and
meet the guidelines for permanence and durability of the Committee
on Production Guidelines for Book Longevity of the Council on
Library Resources

Printed in the United States of America by
Princeton University Press
Princeton, New Jersey

10 9 8 7 6 5 4 3 2 1

And we are then led to our last thought. We *do* think that "admissibility" has some kind of an "if-then" in it. So if what we have heretofore called "extensional admissibility" does not, then what is needed is a new theorem—not just a relevant proof of an old one. Perhaps it would be a new theorem using restricted quantification as described in §80.3.1(4). But, however that comes out, this true relevantist has to look around the logical landscape and say not just that he has not seen a *relevant* proof of the admissibility of the d.s., but that he has not seen even a bad proof of it. Of course such a claim would outrage a classicalist; but Our Hero should not let that bother him.

In any event, we applaud the steadfast courage of the true relevantist as described here. In contrast to the old-fashioned logical empiricists and the new-fashioned nominalists and such, the true relevantist is truly tough-minded, with nary a soft spot in his head. His brow wrinkles, his jaw juts, and he will never, ever use the d.s.

A sober closing: see Lance 1988 for a detailed argument that deep philosophical commitments based on understanding language from a truly social perspective support the use of some form of relevance logic as the *only* viable standard of all reasoning in any social context.

§81. A useful four-valued logic: How a computer should think. The work of the previous section can be understood from a number of points of view. On one of these we can see it as working out a four-valued logic, the values being the various subsets of $\{T, F\}$. We propose that this four-valued logic should sometimes be *used*.

§81.1. The computer. A lot of work has been done recently on applying many-valued logics to the design of computer circuitry and thus giving them application (see Wolf's bibliography in Dunn and Epstein 1977); so what, you may ask, is special about offering a four-valued logic as "useful"? In fact we think we are indeed involved in an odd sort of enterprise; for in the present context we want to use "logic" in a narrow sense, the old sense: "logic" in the sense of an *organon*, a tool, a canon of inference. And it is our impression that hardly any of what individual practitioners of many-valued logic have done is directly concerned with developing logics to use as practical tools for inference. Hence the peculiarity of our task, which is to suggest that a certain four-valued logic ought to be used in certain circumstances as an actual guide to reasoning.

Our suggestion for the utility of a four-valued logic is a local one. It is not the Big Claim that we all ought always to use this logic (unlike the rest of this book, this section does not comment on that claim), but the Small Claim that there are circumstances in which someone—not you—ought to abandon the familiar two-valued logic and use another instead. It will be important to delineate these circumstances with some care.

The situation we have in mind may be described as follows. In the first place, the reasoner who is to use this logic is an artificial information processor, that is, a (programmed) *computer*. This already has an important consequence. People sometimes give as an argument for staying with classical two-valued logic that it is tried and true, which is to say that it is imbued with the warmth of familiarity. This is a good (though not conclusive) argument for anyone who is interested, as we occasionally are, in practicality; it is akin to Quine's principle of "minimal mutilation," though we specifically want the emotional tone surrounding familiarity to be kept firmly in mind. But, given that in the situation we envisage the reasoner is a computer, this argument has no application. The notion of "familiarity to the computer" makes no sense, and surely the computer does not care what logic is familiar to *us*. Nor is it any trouble for a programmer to program an unfamiliar logic into the computer. So much for emotional liberation from two-valued logic.

In the second place, the computer is to be some kind of sophisticated question-answering system, where by "sophisticated" we mean that it does not confine itself, in answering questions, to just the data it has explicitly in its memory banks, but can also answer questions on the basis of *deductions* that it makes from its explicit information. Such sophisticated devices barely exist today, but they are in the forefront of everyone's hopes. In any event, the point is clear: unless there is some need for reasoning, there is hardly a need for logic.

Thirdly, the computer is to be envisioned as obtaining the data on which it is to base its inferences from a variety of sources, all of which may be supposed to be on the whole trustworthy, but none of which can be assumed to be that paragon of paragons, a universal truth-teller. There are at least two possible pictures here. One puts the computer in the context of a lot of fallible humans telling it what is so and what is not, or, with rough equivalency, a single human feeding it information over a stretch of time. The other picture paints the computer as part of a network of artificial intelligences with whom it exchanges information. In any event, the essential feature is that there is no single, monolithic, infallible source of the computer's data, but that inputs come from several independent sources. In such circumstances *the* crucial feature of the situation emerges: *inconsistency threatens*. Elizabeth tells the computer that the Pirates won the Series in 1971; Sam tells it otherwise. What is the computer to do? If it is a classical two-valued logician, it must give up altogether talking about anything to anybody, or, equivalently, it must say everything to everybody. We all know all about the fecundity of contradictions in two-valued logic: contradictions are never isolated, infecting as they do the whole system. Of course the computer could refuse to entertain inconsistent information. But in the first place that is unfair both to Elizabeth and to Sam, each of whose credentials are, by hypothesis, nearly impeccable. And in the second place, as we know all too well, contradictions

may not lie on the surface. There may be in the system an *undetected* contradiction, or, what is just as bad, a contradiction that is not detected until long after the input that generated it has been blended in with the general information of the computer and has lost its separate identity. But still we want the computer to use its head to reason to just conclusions yielding sensible answers to our questions.

Of course we want the computer to report any contradictions that it finds, and in that sense we by no means want the computer to ignore contradictions. It is just that where there is a possibility of inconsistency, we want to set things up so that the computer can continue reasoning in a sensible manner even if there is such an inconsistency, discovered or not. And, even if the computer has discovered and reported an inconsistency in its baseball information, such as that the Pirates both won and did not win the Series in 1971, we would not want that to affect how it answered questions about airline schedules. But if the computer is a two-valued logician, the baseball contradiction will lead it to report that there is no way to get from Bloomington to Chicago. And also, of course, that there are exactly 3,000 flights per day. In an incisive phrase, S. C. Shapiro calls this "polluting the data." What we are proposing is to Keep Our Data Clean. (Shapiro and Wand 1976 and also Shapiro separately have independently argued for the utility of relevance logics for question-answering systems, and have suggested implementation; see §83 for a detailed account.)

So we have a *practical* motive for dealing with situations in which the computer may be told both that a thing is true and that it is false (at the same time, in the same place, in the same respect, etc., etc., etc.).

There is a fourth aspect of the situation, concerning the significance of which we remain uncertain, but which nevertheless needs mentioning for a just appreciation of developments below: our computer is *not* a complete reasoner, who should be able to do something better in the face of contradiction than just report. The complete reasoner should, presumably, have some strategy for *giving up* part of what it believes when it finds its beliefs inconsistent. Since we have never heard of a practical, reasonable, mechanizable strategy for revision of belief in the presence of contradiction, we can hardly be faulted for not providing our computer with such. In the meantime, while others work on this extremely important problem, our computer can only accept and report contradictions without divesting itself of them.

This aspect is bound up with a fifth: in answering its questions, the computer is to reply strictly in terms of what it has been told, *not* in terms of what it could be programmed to believe. For example, if it has been told that the Pirates won and did not win in 1971, it is to so report, even though we could of course program it to recognize the falsity of such a report. The point here is both subtle and obvious: if the computer would not report out contradictions in answer to our questions, we would have no way of knowing that

its data base harbored contradictory information. (We could, if we wished, ask it to give a *supplementary* report, e.g., as follows: "I've been told that the Pirates won and did not win; but of course it ain't so"; but would that be useful?)

Approximation lattices. Always in the background and sometimes in the foreground of what we shall be working out is the notion of an *approximation lattice*, due to Scott 1970b, 1972, 1973a; see the *Compendium* Gierz, Hofmann, Keimel, Lawson, Mislove, Scott 1980. Let us say a word about this concept before getting on. You are going to be disappointed at the mathematical definition of an approximation lattice: mathematically it is just a complete lattice. That is, we have a set A on which there is a partial ordering $\sqsubseteq$, and for arbitrary subsets X of A there always exist least upper bounds $\bigsqcup X \in A$ and greatest lower bounds $\bigsqcap X \in A$ (two-element ones written $x \sqcup y$ and $x \sqcap y$). But we don't call a complete lattice an approximation lattice unless it satisfies a further, nonmathematical condition: it is appropriate to read $x \sqsubseteq y$ as "x approximates y." Examples worked out by Scott include the lattice of "approximate and overdetermined real numbers," where we identify an approximate real number with an interval, and where $x \sqsubseteq y$ just in case $y \subseteq x$. The (only) overdetermined real number is the empty set. As a further example Scott offers the lattice of "approximate and overdetermined functions" from A to B, identified as subsets of $A \times B$. Here we want $f \sqsubseteq g$ just in case $f \subseteq g$.

In such lattices the *directed* sets are important: those sets such that every pair of members x and y of the set have an upper bound z also in the set. For such a set can be thought of as approximating by a limiting process to its union $\bigsqcup X$. That is, if X is directed, it makes sense to think of $\bigsqcup X$ as the limit of X. (An ascending sequence $x_1 \sqsubseteq \ldots \sqsubseteq x_i \sqsubseteq \ldots$ is a special case of a directed set.) And now when we pass to the family of functions from one approximation lattice into another (or of course the same) approximation lattice, Scott has demonstrated that what are important are the *continuous* functions: those which preserve nontrivial directed unions (i.e., $f(\bigsqcup X) = \bigsqcup\{fx: x \in X\}$, for nonempty directed X). These are the only functions that respect the lattices qua *approximation* lattices. This idea is so fundamental to developments below that we choose to catch it in a "thesis" to be thought of as analogous to Church's thesis:

SCOTT'S THESIS. In the presence of complete lattices A and B, naturally thought of as approximation lattices, pay attention only to the continuous functions from A into B, resolutely ignoring all other functions as violating the nature of A and B as approximation lattices.

(Though honesty compels us to attribute the thesis to Scott, the same policy bids us note that the formulation is ours and that, as it is stated, Scott may

not want it or may think that some other thesis in the neighborhood is more important, for example, that every approximation lattice (intuitive sense) is a continuous lattice (sense of Scott 1972a).)

You will see how we rely on Scott's thesis in what follows.

Program. The rest of this section is divided into three parts. Part 1 (§81.2) considers the case in which the computer accepts only *atomic* information. This is a heavy limitation, but provides a relatively simple context in which to develop some of the key ideas. Part 2 (§81.3) allows the computer to accept also information conveyed by *truth-functionally compounded sentences*; and in this context we offer a new kind of meaning for formulas as certain mappings from epistemic states into epistemic states. In Part 3 (§81.4) the computer is allowed also to accept *implications construed as rules* for improving its data base.

§81.2. Part 1. Atomic inputs. We first consider the computer receiving only "simple" or atomic bits of information on the basis of which to answer (possibly complex) questions.

§81.2.1. Atomic sentences and the approximation lattice A4. Now and throughout this paper you must keep firmly fixed in mind the circumstances in which the computer finds itself, and especially that it must be prepared to receive and reason about inconsistent information. We want to suggest a natural technique for employment in such cases: when an item comes in as asserted, mark it with a "told True" sign, and when an item comes in denied, mark with a "told False" sign, treating these two kinds of tellings as altogether on a par. In a phrase we have used elsewhere, this is a "double-entry bookkeeping" and it is easy to see that it leads to four possibilities. For each item in its basic data file, the computer is going to have it marked in one of the following four ways: (1) just the "told True" sign, indicating that that item has been asserted to the computer without ever having been denied; (2) just the value "told False," which indicates that the item has been denied but never asserted; (3) no "told" values at all, which means the computer is in ignorance, has been told nothing; (4) the interesting case: the item is marked with both "told True" and "told False." (Recall that allowing this case is a *practical necessity* because of human fallibility.)

These four possibilities are precisely the four values of the many-valued logic we are offering as a practical guide to reasoning by the computer. Let us give them names:

T: just told True (warning: not same as ***T*** of §50 and elsewhere)
F: just told False (ditto)
None: told neither True nor False
Both: told both True and False

So these are our four values, and we baptize: **4** = {***T***, ***F***, ***None***, and ***Both***}. Of course four values do not a logic make, but let us nevertheless pause a minute to see what we have so far.

The suggestion requires that a system using this logic code each of the atomic statements representing its data base in some manner indicating which of the four values it has (at the present stage). It follows that the computer cannot represent a class merely by listing certain elements, with the assumption that those not listed are not in the class. For, just as there are four values, so there are four possible states of each element: the computer might have been told none, one, or both of "in the class" and "not in the class." Two procedures suggest themselves. The first is to list each item with one of the values ***T***, ***F***, or ***Both***, for these are the elements about which the computer has been told something; and to let an absence of a listing signify ***None***, i.e., that there is no information about that element. The second procedure would be to list each element with one *or both* of the "told" values, "told True" and "told False," not listing elements lacking both "told" values. This amounts to the §50 *relations* of formulas to (told) truth values. Obviously the procedures are equivalent, and we shall not in our discourse distinguish between them, using one or the other as seems convenient.

The same procedure works for relations, except that it is ordered pairs that get marked. For example, a part of the correct table for Series winners, conceived as a relation between teams and years, might look like this:

⟨Pirates, 1971⟩ ***T***	or	⟨Pirates, 1971⟩ True
⟨Orioles, 1971⟩ ***F***		⟨Orioles, 1971⟩ False

But if Sam slipped up and gave the wrong information after Elizabeth had previously entered the above, the first entry would become

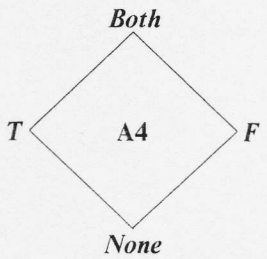
⟨Pirates, 1971⟩ ***Both*** or ⟨Pirates, 1971⟩ True, False

To be specific, we envision (in this Part of the section) the epistemic state of the computer to be maintained in terms of a table giving one of four values to each atomic sentence. We call such a table a *set-up* (following an isomorphic use of Routley and Routley 1972; see §16.2.1); i.e., a set-up is, mathematically, a mapping from atomic sentences into the set **4** = {***T***, ***F***, ***None***, ***Both***}. When an atomic formula is entered into the computer as either affirmed or denied, the computer modifies its current set-up by adding a "told True" or "told False" according as the formula was affirmed or denied; it does not subtract any information it already has, for that is the whole point of what we are up to. In other words, if p is affirmed, it marks p with ***T*** if p was previously marked with ***None***, with ***Both*** if p was previously marked with ***False***; and of course leaves things alone if p was already marked either ***T*** or ***Both***. So much for p as input.

The computer not only accepts input, but answers questions. We consider only the basic question as to p; this it answers in one of four ways: Yes, No, Yes and No, or I don't know, depending on the value of p in its current set-up as ***T***, ***F***, ***Both***, or ***None***. (It would be wrong to suppose that these four answers are either dictated by the four-valued logic or excluded by the two-valued logic; it is just that they are made more useful in the four-valued context. See Belnap 1963 and Belnap and Steel 1975.)

Warning—or, as N. Bourbaki says, *tournant dangereux* (∞): "told True" is *not* equivalent to ***T***. The relationships are rather as follows. In the first place, the computer is told True about some sentence A just in case it has marked A *either* with ***T*** *or* with ***Both***. Secondly, the computer marks A with ***T*** just in case it has been told True of A *and* has not been told False of A. And similarly for the relation between ***F*** and "told False." These relationships are certainly obvious, but also in practice confusing. It might help always to read "told True" as "told *at least* True," and ***T*** as "told *exactly* True."

We now make the observation that consitutes the foundation of what follows: these four values naturally form a lattice under the lattice-ordering "approximates the information in"; indeed they form an *approximation lattice* in the sense we described above:



In this Hasse diagram joins ($\sqcup$) and meets ($\sqcap$) are least upper bounds and greatest lower bounds, respectively, and $\sqsubseteq$ goes uphill. ***None*** is at the bottom because it gives no information at all; and ***Both*** is at the top because it gives too much (inconsistent) information, saying as it does that the statement so marked is held both told True/told False. As we mentioned above, Scott has studied approximation lattices in detail and in a much richer setting than we have before us; yet still this little four-element lattice is important for much of his work. We remarked above that, according to Scott's thesis, the important functions in the neighborhood of an approximation lattice like **A4** are the continuous ones. We do not, fortunately, have to deal with continuity for a while, since in the finite case it turns out that for a function f to be continuous is just for it to be *monotonic*, i.e., for it to preserve the lattice ordering: $a \sqsubseteq b$ implies $fa \sqsubseteq fb$.

For example, suppose a function g on **A4** is such that it takes ***T*** into ***F*** and ***F*** into ***T***: $g(\boldsymbol{T}) = \boldsymbol{F}$, $g(\boldsymbol{F}) = \boldsymbol{T}$. Then, given that g is monotonic, since

$T \sqsubseteq$ ***Both*** we must have $F \sqsubseteq$ g(***Both***) and similarly $T \sqsubseteq$ g(***Both***). So we must have g(***Both***) = ***Both***. In a similar way, it is easy to calculate that g(***None***) = ***None***—if g is to be monotonic, as all good functions should be.

§81.2.2. Compound sentences and the logical lattice L4. Now this function g is no *mere* example of a monotonic function on the lattice **A4** of approximate and inconsistent truth values. In fact we are in the very presence of *negation*, which some have called the original sin of logic, but which we clearly need in a sufficiently rich language for our computer to use—just to be able to answer simple yes–no questions. To see that g really *is* negation, consider first that the values ***T*** and ***F***, representing as they do the pure case, should act like the ordinary truth values the True and the False; so obviously we want $\sim T = F$, and $\sim F = T$. And then Scott's thesis now imposes on us a *unique* solution to the problem of extending negation to the values of our foursome; we *must* have $\sim$***None*** = ***None*** and $\sim$***Both*** = ***Both*** if negation is to be an acceptably monotonic function on the approximation lattice **A4**.

We can summarize the argument in a small table for negation.

	None	***F***	***T***	***Both***
	m	tt	tt	m
$\sim$	***None***	***T***	***F***	***Both***

Here "tt" in the upper right-hand corner means that the value was given by truth-table considerations, and "m" indicates that monotonicity was invoked.

Having put negation in our pocket, let's turn to conjunction and the disjunction. We start with truth-table considerations for the ***T***–***F*** portion of the tables, and then invoke monotony (in each argument place) and easy considerations to extend them as indicated.

&	***None***	***F***	***T***	***Both***
	m		m	
None	***None***		***None***	
		tt	tt	
F		***F***	***F***	
	m	tt	tt	m
T	***None***	***F***	***T***	***Both***
			m	m
Both			***Both***	***Both***

$\vee$	*None*	*F*	*T*	*Both*
None	m *None*	m *None*		
F	m *None*	tt *F*	tt *T*	m *Both*
T		tt *T*	tt *T*	
Both		m *Both*		m *Both*

With just ordinary truth tables and monotonicity, it would appear we have to stop with these partial tables; on this basis neither conjunction nor disjunction—unlike negation—is uniquely determined. Of course we might make some guesses on the basis of intuition, but this part of the argument is founded on a desire not to do that; rather, we are trying to see how far we can go on a purely theoretical basis.

It turns out that, if we ask only that conjunction and disjunction have some minimal relation to each other, then every other box is uniquely determined. There are several approaches possible here, but perhaps as illuminating as any is to insist that the orderings determined by the two in the standard way be the same; which is to say that the following equivalence (see e.g., the end of §28.2.1.) holds:

$$a\&b = a \quad \text{iff} \quad a \vee b = b$$
$$a\&b = b \quad \text{iff} \quad a \vee b = a$$

For look at the partial table for conjunction. One can see that $\boldsymbol{T}$ is an identity element: $a\&\boldsymbol{T} = a$, for all a. So, if conjunction and disjunction fit together as they ought, we must have $a \vee \boldsymbol{T} = \boldsymbol{T}$, for all a, which fills in two boxes of the $\vee$-table. And similar arguments fill in all except the corners.

For the corners we must invoke monotonicity (*after* the above lattice argument). For example, since $\boldsymbol{F} \sqsubseteq \boldsymbol{Both}$, by monotonicity $(\boldsymbol{F} \,\&\, \boldsymbol{None}) \sqsubseteq (\boldsymbol{Both} \,\&\, \boldsymbol{None})$, so $\boldsymbol{F} \sqsubseteq (\boldsymbol{Both} \,\&\, \boldsymbol{None})$. Similarly, $\boldsymbol{None} \sqsubseteq \boldsymbol{F}$ leads to $(\boldsymbol{Both} \,\&\, \boldsymbol{None}) \sqsubseteq (\boldsymbol{Both} \,\&\, \boldsymbol{F})$, i.e., $(\boldsymbol{Both} \,\&\, \boldsymbol{None}) \sqsubseteq \boldsymbol{F}$. So, by antisymmetry in **A4**, $(\boldsymbol{Both} \,\&\, \boldsymbol{None}) = \boldsymbol{F}$. These additional results are brought together in the fol-

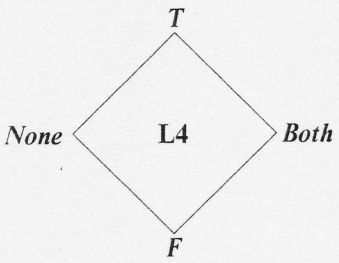
lowing tables, where "f" indicates use of the above fit between & and ∨, and "m" again indicates monotonicity.

&	*None*	*F*	*T*	*Both*
None	*None*	f *F*	*None*	m *F*
F	f *F*	*F*	*F*	f *F*
T	*None*	*F*	*T*	*Both*
Both	m *F*	f *F*	*Both*	*Both*

∨	*None*	*F*	*T*	*Both*
None	*None*	*None*	f *T*	m *T*
F	*None*	*F*	*T*	*Both*
T	f *T*	*T*	*T*	f *T*
Both	m *T*	*Both*	f *T*	*Both*

We don't know whether we should be surprised or not, but in fact these tables (isomorphic to Smiley's matrix of §15.3, which is their historical as opposed to theoretical source) do constitute a lattice, with conjunction as

meet and disjunction as join; a lattice which can be pictured as follows:



Let us agree to call this the *logical* lattice **L4** (it is the SL of §34.1), to distinguish it from the *approximation* lattice **A4**. The ordering on **L4** we write as $a \leq b$; we write meets as a&b, and joins as $a \vee b$. We note that in the logical lattice, each of the values ***None*** and ***Both*** is intermediate between ***F*** and ***T***; and this is as it should be, for the worst thing to be told is that something you cling to is false, *simpliciter*. You are better off (it is one of your hopes) either being told nothing about it or being told both that it is true and also that it is false; while of course best of all is to be told it is true, with no muddying of the waters. Nevertheless, surely most of you must be puzzled, if you are thinking about it, concerning the rules for computing the conjunction and disjunction of ***None*** and ***Both***: (***None*** & ***Both***) = ***F***, while (***None*** $\vee$ ***Both***) = ***T***. We ask you for now only to observe that we were driven to these equations by only three considerations: ordinary truth tables, monotonicity, and fit between & and $\vee$. But we shall have more to say about this.

We can now use these logical operations on **L4** to induce a semantics for a language involving &, $\vee$, and $\sim$, in just the usual way. Given an arbitrary set-up s—a mapping, you will recall, of atomic formulas into **4**—we can extend s to a mapping of *all* formulas into **4** in the standard inductive way:

$$\begin{aligned} s(A\&B) &= s(A)\&s(B) \\ s(A \vee B) &= s(A) \vee s(B) \\ s(\sim A) &= \sim s(A) \end{aligned}$$

And this tells us how the computer should answer questions about complex formulas based on a set-up representing its epistemic state (what it has been told): just as it does for answering questions about atomic formulas, it should answer a question as to A by Yes, No, Yes and No, or I don't know, according as the value of A in s (i.e., s(A)) is ***T***, ***F***, ***Both***, or ***None***.

The preceding discussion will have struck you as abstractly theoretical; we should next like to take up negation, conjunction, and disjunction from an altogether different and more intuitive point of view. The question to which

we are going to address ourselves is this: Given our intuitive understanding of the meaning of the four truth values as indicating markings of sentences with either or both of the True and the False, what is a plausible way to extend these values to compound sentences when we know the values of the ingredient sentences?

In the context of our enterprise, we can sharpen this question and distinguish it from others. We are *not* asking, What is meant by the truth-functional connectives of English—or any language, informal or formal? Rather, we are simply asking, How do *we* want our computer to answer *our* questions about compound sentences, given that we have decided how it is supposed to answer our simpler questions? The question presupposes something nontrivial: namely, that in fact we want a functional relationship between how the computer answers questions about the parts and how it answers questions about the whole when its input is entirely atomic. The reason for this presupposition? We think it tends to simplify our dealings with the computer by increasing our intellectual control over what we ourselves have created, and, furthermore, it is likely to be more easily and efficiently managed by the computer than some other than truth-functional alternatives. But it is not an article of faith.

Let us take up negation first. The inevitable thing to say seems to be that $\sim A$ should be marked "told True" just in case A is marked "told False," and vice versa. In other words, we want the computer to answer "Yes" to "$\sim A$?" just in case it answers "No" to "A?" and vice versa. But then consider the correspondences:

None: marked with neither
F: marked with just told False
T: marked with just told True
Both: marked with both.

It immediately comes out that we should mark $\sim A$ with ***Both*** if A is, with ***None*** if A is, and with ***T*** or ***F*** if A is ***F*** or ***T***. For example, if A is marked ***None***, i.e., with neither told True nor told False, then $\sim A$ should also be marked with neither. If you know nothing about A, then you know nothing about $\sim A$. And the same reasoning works for ***Both***: if you know too much about A, then you also know too much about $\sim A$.

In a similar way, we can give intuitive clauses for evaluation of conjunctions and disjunctions, as follows:

Mark $(A\&B)$ with at least told True just in case both A and B have been marked with at least told True.
Mark $(A\&B)$ with at least told False just in case at least one of A and B have been marked with at least told False.

This completely determines how to mark conjunctions.

Mark $(A \vee B)$ with at least told True just in case at least one of A and B have been marked with at least told True.

Mark $(A \vee B)$ with at least told False just in case both A and B have been marked with at least told False.

And this similarly uniquely determines disjunction, given our intuitive correspondence between our four values ***None***, ***F***, ***T***, ***Both*** on the one hand, and markings with neither, one, or both of told True and told False on the other.

There is another way to read this table, which we give only for one case: we are declaring that we want the computer to answer "Yes" to "$(A \vee B)$?" just in case it is prepared to answer "Yes" to either "A?" or "B?" And we note that this is plausible *only* when, as at present, all input is atomic, so that all disjunctions are decided. In general, commencing with Part 2, we might often expect that the computer can tell us that it was told "$A \vee B$" when it wasn't told either A or B: "told" does not in general distribute over disjunction. But it does in the special, present case, when the entire epistemic state is to be thought of as carried by a set-up.

This intuitive "double-entry bookkeeping" account of the connectives is exactly that of §50.3 with regard to its structure. What we can now go on to observe is that the intuitive account exactly agrees with the theoretically based account deriving from Scott's approximation lattices. For example, consider one of the odd corners, (***Both*** & ***None***) = ***F***. Well, suppose A has been marked both told True and told False, and B with neither (corresponding to ***Both*** and ***None***, respectively). Then the computer must mark $(A\&B)$ at least told False, since one of its components is marked at least told False; and it must *not* mark it at least told True, since not both of its components are so marked (assuming atomic input only). So we must mark it exactly told False. So (***Both*** & ***None***) = ***F***. In other even more informal words, in this circumstance the computer has a reason to suppose $(A\&B)$ told False, but none to suppose it told True. So, although the oddity of (***Both*** & ***None***) = ***F*** doesn't go away, it anyhow gets explained.

§81.2.3. Entailment and inference: The four-valued logic. Where are we? Well, we haven't got a logic, i.e., rules for generating and evaluating inferences. (In our case we really want the former; we want some rules for the computer to use in generating what it implicitly knows from what it explicitly knows.) What we do have is four interesting values, with indications as to how these are to be used by friend computer, and three splendid connectives, with complete and well-motivated tables for each. And, as we all know, lots of other connectives can be defined in terms of these; so for our purposes three is enough.

Suppose we have an argument involving these connectives. The question is, when is it a good one? Again we want to give an abstractly theoretical

answer, and then an intuitive answer. (And then several more answers, too, if there is time enough. For the question is fascinating.)

The abstract answer relies on the *logical* lattice we took so much time to develop. It is: entailment goes uphill. That is, given any sentences A and B (compounded from variables by negation, conjunction, and disjunction), we will say that A *entails* or *implies* B just in case for each assignment of one of the four values to the variables, the value of A does not exceed (is less-than-or-equal-to) the value of B. In symbols: $s(A) \leq s(B)$ for each set-up s. This is a plausible definition of entailment whenever we have a lattice of values that we can think of as somehow being graded from bottom to top; and, as we suggested when first presenting you with the logical lattice, we can indeed think of ***None*** and ***Both*** as being intermediate between awful ***F*** and wonderful ***T***.

Now for an account which is close to the informal considerations underlying our understanding of the four values as keeping track of markings with told True and told False: say that the inference from A to B is valid, or that A entails B, if the inference never leads us from told True to the absence of told True (preserves Truth), *and also* never leads us from the absence of told False to told False (preserves non-Falsity). Given our system of markings, to ask this is hardly to ask too much.

(We note that in §50.6 we have shown that it suffices to mention truth preservation, since if some inference form fails always to preserve non-Falsity, then it *also* fails to preserve Truth. But, as we suggested in §50.6, the False really is on all fours with the True; so it is profoundly natural to state our account of "valid" or "acceptable" inference in a way that is neutral with respect to the two.)

Finally we have a logic, that is, a canon of inference, for our computer to use in making inferences involving conjunction, negation, and disjunction, as well of course as whatever can be defined in terms thereof. We note that this logic has two key features. In the first and most important place, it is rooted in reality. We gave reasons why it would be good for our computer to think in terms of our four values, and why the logic of the four values should be as it is. In the second place, though we have not thrown around many hen-scratches, it is clear that our account of validity is mathematically rigorous. And obviously the computer can decide by running through a truth-tabular computation whether or not a proposed inference is valid. But there is another side to the logician's job, which is codifying inferences in some axiomatic or semi-axiomatic way that is transparent and accordingly usable. If this sounds mysterious, it is not; we just mean that a logician, given a semantics, ordinarily tries to come up with proof theory for it; a proof theory that is consistent and complete relative to the semantics.

Fortunately the job has already been done. It will come as no surprise to anyone, other than the odd reader who elected to read this section first, that

we have given still another characterization of the tautological entailments—the system $\mathbf{E}_{\mathrm{fde}}$—of §15.1. And although we cannot help taking note of this additional evidence for the stability of $\mathbf{E}_{\mathrm{fde}}$, our interests in this section lie in altogether different directions.

§81.2.4. Observations. Some observations now need to be made before pushing further.

First, we note that not derivable from these principles, and not semantically valid, are the paradoxes of "implication" $A\&\sim A \rightarrow B$ and $A \rightarrow B\vee\sim B$. In context, the failure of these principles is evident. The failure of the first simply means that just because we have been told both that A is True and that A is False, we cannot conclude: everything. Indeed, we may have been told nothing about B, or just that it is False. And the failure of the second is equally evident: from the fact that we have been told that A is True, we cannot conclude that we know something about B. Of course B is *ontologically* either True or False, and such ontological truth values will receive their due; but for $B\vee\sim B$ to be marked told True is either for B to be marked told True or for B to be marked told False; and it may have neither mark. Or, for a different way of counterexampling $A \rightarrow B\vee\sim B$, A may have just been told True while $(B\vee\sim B)$ has *both* values because B does.

These inferences are not wanted in a scheme that is designed not to break down in the presence of "contradictions"; and since contradictions really do threaten in the circumstances we describe, their absence is welcome.

We would be less than open, however, if we failed to point out the absence of what at first sight looks like a more harmless principle: our old friend (γ), $(A\vee B)\&\sim A \rightarrow B$. Surely, one would think, our computer should be able to argue that if it is told that one of A and B is True, and it is told that A is False, then it must have been told that B is True. That's true; unless—and of course this is a critical "unless"—there is an inconsistency around. In fact the inference that the canon allows is just exactly

$$(A\vee B)\&\sim A \rightarrow (A\&\sim A)\vee B.$$

That is, having determined that the antecedent is at least told True, we allow the computer to conclude: either B is at least told True, or something funny is going on; i.e., it's been told that A is both True and False. And this, you will see, is right on target. If the *reason* that $(A\vee B)\&\sim A$ is getting thought of as a Truth is because A has been labeled as both told True and told False, then we certainly do *not* want to go around inferring B. The inference is wholly inappropriate in a context where inconsistency is a live possibility.

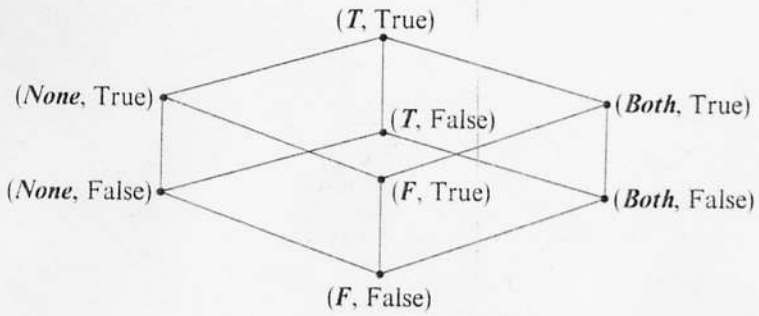
The second observation is that our four values are proposed only in connection with *inferences*, and are definitely not supposed to be used for determining which formulas in $\&$, $\vee$, and $\sim$ count as so-called "logical truths." In fact, no formula takes *always* the value *T*; so *that* property surely won't do as a

semantic account of logical truth. There are, on the other hand, formulas that never take the value ***F***, e.g., $A \vee \sim A$; but this set is not even closed under conjunction and does not contain $(A \vee \sim A)\&(B \vee \sim B)$, which can take ***F*** when A takes ***None*** and B takes ***Both***. So just don't try to base logical truth on these values.

Thirdly, let us consider ontology versus epistemology. One of the difficulties that often arise in relating many-valued logics to real concerns is that one tends to vacillate between reading the various values as epistemic on the one hand, and ontological on the other. Does Łukasiewicz's middle value, $\frac{1}{2}$, mean "doesn't have a proper truth value," or does it mean "truth value unknown"? In informal explanations of what is going on, logicians sometimes move from one of these readings to the other in order to save the interest of the enterprise.

Our four values are unabashedly epistemic. According to our instructions, sentences are to be marked with either a ***T*** or an ***F***, a ***None*** or a ***Both***, according to what the computer has been told; or, with only a slight (but dangerous) metaphor, according to what it believes or knows. Does this somehow make the enterprise wrong-headed? Or not logic? No. Of course these sentences *have* truth values independent of what the computer has been told; but who can gainsay that the computer cannot *use* the actual truth value of the sentences in which it is interested? All it can possibly *use* as a basis for inference is what it knows or believes, i.e., what it has been told.

But we can do better than this. Let us get the ontology into the act by *splitting* our four epistemic values into two, one representing the case in which the sentence is ontologically true, the other the case in which it is false. Obviously we then get eight values instead of four, each of which we may visualize as an ordered pair, the left entry of which is an epistemic value ***T***, ***F***, ***None***, or ***Both***, while the right entry is one of Frege's ontological values the True and the False. Giving the usual classical two-valued tables to the connectives, and also and equivalently, interpreting the implicative connective in the usual way, we are led to the following lattice picture (this is *not* an approximation lattice; it is isomorphic to M_0 of §18.4 and elsewhere):



The &s and ∨s can be computed, respectively, as greatest lower bounds and least upper bounds, while negation-pairs are: two left, two center, two right, and top-bottom (*not* the Boolean way). The values of this new many-valued logic have a mixed status: they are in part epistemological and in part ontological. Should we then move to this logic? It is entertaining to observe that there is no need to do so for inferences; for *exactly* the same inferences are valid with this as with our four-valued canon of inference. Nor should this be surprising, for two reasons. In the first place, as we already observed, the only things we can actually *use* in inference are the epistemic values ***T***, ***F***, ***None***, and ***Both***, representing what we know, believe, or in any event have been told by authority we by and large trust. Secondly, and more prosaically, observe that all the inferences sanctioned by the four-valued canon are already approved in two-valued logic; so adding as a condition that ontological truth is to be preserved is to add a condition that is already satisfied and yields no new constraints. So for practical reasons there is no need to move from four to eight values for judging inferences. In the words of a famous philosopher, "Do not multiply many values beyond necessity."

If, however, for some reason (we do not just now know what) someone wanted an account of logical truth in &, ∨, and ∼, then one could invoke as a criterion: being always True (by the right entry of the pair) regardless of what you've been told (according to the left entry). Then, not surprisingly, one finds out that the two-valued tautologies are precisely the logical truths on this account. Not surprisingly, because we invoke values only ontological, throwing away (in the eight-valued case) all the information of the epistemic values.

Let us say explicitly, if it is not obvious, that we think this codification of truth-functional logical truths not important to the computer; for what was *wanted* was a way of *reasoning* from and to truth-functional compounds, not a sorting of these compounds.

Our fourth "observation" is not so much an observation as it is an inconclusive discussion of the role of ***Both*** and ***None***. The problem is that one is inclined to the view that they should be identified, that the computer is in the same state having been told that A is both True and False as it is having been told nothing about A. Both S. Haack and A. Kenny have each, in quite different ways, suggested something like this. If you will be satisfied with a dialectical flourish, we can supply one of the form "Wrong, but understandable." It goes like this. In the first place, it is somehow magnificently obvious that ***Both*** and ***None*** should not be identified, as H. S. Harris noted in conversation, just because we want the computer to distinguish *for us* when it has been told a contradiction from when it has been told nothing. This is surely essential on anyone's view. In the second, our developments can be taken as explaining the feeling that they should be identified, for just look at the logical lattice **L4**: there ***Both*** and ***None*** occupy (distinct but) absolutely sym-

metrical positions between ***F*** and ***T***, and in this sense are "identified." For instance, we allow the inference from neither to ***F***, and to neither from ***T***, and thus treat them alike.

Still, though this response may be helpful, we are not altogether happy with it. And we much prefer to leave the discussion as at this stage incomplete.

Our penultimate observation concerns the suggestion that the computer keep more information than we have allowed it to keep. Perhaps it should count the number of times it has been told True or told False, or perhaps it should keep track of its sources by always marking, for example, "told True by Sam at 2200:03 on 4 August 1973." We do not see why these ideas should not be explored, but two comments are in order. The first is that it is by no means self-evident how this extra information is to be utilized in answering questions, in inference, and in the input of complex sentences. That is, one should not be misled by the transparency of the idea in the case of atomic sentences. The consequence of this first comment is merely that the exploration lies ahead. The second is the practical remark that there are severe costs in carrying extra information, costs which may or may not be worth incurring. And if there are circumstances in which they are not worth incurring, we are back to the situation we originally described.

Lastly, we want to mention some alternatives without (much) discussing them. Gupta has noted that one could define the value of A in s not directly as we have done, but rather by reference to all the consistent sub-set-ups of s. Definitions: s′ is a *sub-set-up* of s if it approximates it: $s' \sqsubseteq s$. And s′ is *consistent* if it never awards ***Both***. Finally, let $s(A)$ be defined, in Gupta's way, by $s(A) = \bigsqcup \{s'(A)$: s′ is a consistent sub-set-up of s$\}$, where $s'(A)$ is as already defined. The idea is clearly dual to van Fraassen's 1969a definition of true-in-a-valuation by reference to all the complete (i.e., all truth-value gaps filled) "supervaluations" of a given valuation. One notes that if $s(p) =$ ***Both*** then the question as to p (on s) will be answered "Yes and No" as before, while the question as to $p \& \sim p$ will be answered just "No," instead of "Yes and No."

A related idea is to follow van Fraassen 1969a directly by looking at all the complete super-set-ups of a given set-up; this would give always "Yes" to $p \vee \sim p$. And carrying this idea to its logical conclusion would combine the two ideas (if possible).

All these things are possible. One would hope, however, that the discussion of the alternatives would circulate around the question, How in fact do we want the computer to answer our questions? Thus they would not be *mere* possibilities.

Quantifiers. Quantifiers introduce a number of subtleties to which we shall merely tip our hat, while recognizing that treating them in detail is quite essential to our enterprise.

There is in the first place the question of whether "the" domain is finite or infinite. Both cases can plausibly arise. In the latter case, there is a question of how the computer is to represent infinite information with its finite resources, but one should not infer from the existence of this problem that the computer can't or shouldn't involve itself with quantification over infinite domains. Surely it should be allowed to answer "Is there a number such that . . . ?" queries (if it can).

In the second place, there is the question of whether the computer has a name for everything in "the" domain, so that we can employ the substitutional interpretation of the quantifiers, or, on the other hand, does not have a name for each entity in "the" domain, so that the domain-and-values interpretation is forced. Again: both cases can plausibly arise, though attending to standard examples like baseball queries or airline flights might have made one think that in the computer situation everything always has a name. But, for example, in some of Isner's work (1975) the computer is told "there is something between a and b" in a context in which it hasn't got a complete list of either the names or the entities against which to interpret this statement. And still it must work out the consequences, and answer the questions it is given. (Of course it is OK for the computer to make up its own name for the "something" between a and b; but that is both an important and an entirely different matter.)

In any event, the semantics given for the connectives extend to universal and existential quantifiers in an obvious way, and we suppose the job done. And the various alternatives mentioned above turn out not to make any difference to the *logic* (with the obvious exception of the finite everything-has-a-name case): the valid "first degree entailments" of §40 do admirably (supplemented, in the finite case, with the principle that a conjunction that runs through the domain implies the appropriate universal statement).

§81.3. Part 2. Compound truth-functional inputs. We can pause now if we like with regard to the overall title of this section, for it would be possible to do so and still claim the title appropriate: we really have presented a four-valued logic and argued that it is useful. But there is a fair bit more to do, some of it of theoretical interest, some of it practical. We begin with considerations closer to the practical.

§81.3.1. Epistemic states. So far, in Part 1, we have been considering the situation in which the epistemic state of the computer could be represented by tables specifying for the various atomic formulas which of the four values in **4** each is to take. We called the mathematical equivalent of such a table a *set-up*; that is, a set-up s is a mapping from all atomic formulas into **4**: $s(p) \in \mathbf{4}$. Let S be the set of all set-ups, and recall that each $s \in S$ extends uniquely to map all formulas into **4**: $s(A) \in \mathbf{4}$.

Each set-up s represents (not what is true but) what the computer has been told. But can every epistemic state of the computer be represented by a set-up? If in fact, as in Part 1, only atomic sentences are affirmed or denied to the computer, of course; but not in general otherwise. For example, no single set-up can represent the state the computer should be in when it is told that either P, the Pirates, or O, the Orioles, won in 1971, but it isn't told which. Set-ups can, by judicious use of ***None***, represent some kinds of incomplete information, but not this kind. For any single set-up in which "either P or O" (with obvious meaning) is marked told True is a set-up in which either P or O is *also* marked told True and, hence, has too much information. Any such set-up would lead the computer to answer "Yes" either to the question, Did the Pirates win? or to the question, Did the Orioles win? And the computer should not be able to answer either of these questions, having been told only that either the Pirates or the Orioles won.

The solution to this problem is well known in the logical literature, going back to Carnap 1942 at least. It has been used in epistemic and doxastic logic by Hintikka 1962 and has also been worked out for computers by Isner 1972, 1975: one uses a *collection* of set-ups to represent a single epistemic state, the rough and partial idea being that the computer takes a formula as something it has been told if it comes out told True on *each* of the set-ups forming its current epistemic state. For example, when told that either the Pirates or the Orioles won, the computer would represent this information by building *two* set-ups, one in which the Pirates get ***T*** and the Orioles ***None***, and the other in which the Orioles get ***T*** and the Pirates ***None***. Later, when it is asked whether the Pirates won, it will say that it doesn't know, since the Pirates are not marked ***T*** in every state, and also not ***F*** in every state; and similarly if it is asked about the Orioles. But if it is asked "Did either the Pirates or the Orioles win?" then it will answer affirmatively, since that sentence is marked told True in both of the set-ups in its epistemic state.

Let us, therefore, at least for the duration of this Part, define an *epistemic state* as a nonempty collection of set-ups, a (nonempty) subset, that is, of S. (If we later omit "nonempty," please supply it, or identify the empty set with the unit set of the set-up that marks everything in sight with ***Both***.) We let ES be the set of all epistemic states, and use "E" as ranging over ES. Let E be an epistemic state. Then the "meaning" of E is that the computer has been told that the world is accurately (but perhaps incompletely) described by at least one of the set-ups in E. As from the beginning the possibility exists that such a description is inconsistent.

E represents the basis on which we want the computer to answer our questions. And let us now state more completely and more accurately how we want our questions answered, by defining the value of a sentence in an epistemic state; in symbols, E(A) for E $\in$ ES and A a formula. Note how the key idea of approximation is mobilized to give insight into what is going

on: the value of a sentence in an epistemic state is to be determined by taking the *meet* of all its values in the separate states—the meet to be taken not in the logical lattice **L4** but in the approximation lattice **A4**. In notation:

$$\mathrm{E}(A) = \sqcap \{\mathrm{s}(A)\colon \mathrm{s}\in\mathrm{E}\}.$$

The idea of this definition is straightforward and intuitively appealing. In the first place, we noted that set-ups individually tend to give us more information than we've got about a formula, or in the language of approximation,

$$\mathrm{E}(A) \sqsubseteq s(A) \qquad \text{for all } \mathrm{s}\in\mathrm{E}.$$

Now what we are saying is that E(A) should be defined so as to be *maximal* while retaining this relationship; i.e., E(A)—the value of A in E—should be the *greatest* lower bound of all the s(A) for s∈E.

EXAMPLE. Let E = {s, s′}, where

$$\left.\begin{array}{ll} \mathrm{s(P)} = \boldsymbol{T} & \mathrm{s'(P)} = \boldsymbol{None} \\ \mathrm{s(O)} = \boldsymbol{None} & \mathrm{s'(O)} = \boldsymbol{T} \\ \mathrm{s(B)} = \boldsymbol{T} & \mathrm{s'(B)} = \boldsymbol{F} \\ \mathrm{s(M)} = \boldsymbol{T} & \mathrm{s'(M)} = \boldsymbol{Both} \end{array}\right\} \quad \text{then:} \quad \left\{\begin{array}{l} \mathrm{E(P)} = \boldsymbol{None} \\ \mathrm{E(O)} = \boldsymbol{None} \\ \mathrm{E(B)} = \boldsymbol{None} \\ \mathrm{E(M)} = \boldsymbol{T} \end{array}\right.$$

Further, though E(P) = E(O) = ***None***, clearly E($P \vee$ O) = ***T***.

Let us, as usual, relate this to marking told True and told False: it all amounts to saying that we should mark A told True in E if it is marked told True in all set-ups in E, and mark it told False if it is marked told False in all set-ups in E; recognizing, as always, that this recipe allows marking A with neither or both. On this account the similarities emerge to van Fraassen 1969a's supervaluations, to definitions of necessity and impossibility in modal logic (e.g., Kripke 1963), and to evaluation of epistemic operators in Hintikka 1962. But of course in all those cases set-ups are restricted to those which are consistent, nor is there any sense in which any of those logics are four-valued, or even three-valued. (Van Fraassen's formulas can take three values, but the third does not have a *logical* relation to the other two, nor is his semantics truth-functional.)

In now extending the account of question-answering, again treat only the simple question as to A in the context of an epistemic state E. It goes just as before: the computer answers Yes, No, Yes and No, or I don't know, according as the value E(A) of A in E is ***T***, ***F***, ***Both***, or ***None***. If, for example, the value of A in its current state is ***Both***, the computer answers "Yes and No." Of course, in this case the asker of the question will know that the answer is based on an inconsistency—and so will the computer. Indeed, this is how the computer would naturally *report* an inconsistency in an epistemic state; recall that the answer does not have the ontological force of "That's the way

the world is," but rather has the epistemic force of "That's what I've been told (by people I trust to get it generally right)."

There are at least three situations in which the computer has to deal with formulas: when asked a question, as we have just discussed; when calculating or inferring, which we have discussed some and to which we shall return, and when a formula is input. It is this last which is now up for discussion, but further developments are going to be easier if some additional approximation lattices are introduced at this point.

§81.3.2. More approximation lattices. Note first that the family S of all set-ups constitutes a natural approximation lattice AS, where the order is pointwise:

$s \sqsubseteq s'$ iff, for each atomic sentence p, $s(p) \sqsubseteq s'(p)$ (in **A4**).

That is, one set-up approximates another if, for each atomic formula p, the information the first set-up gives about p approximates the information the other set-up gives about p. Our point is not only that AS is a complete lattice (we need that mathematically), but that it is *natural* to interpret its ordering as an approximation: if one increases the information on one of the atomic formulas, one increases the information in the set-up.

Since AS is infinite, for the first time in the course of these deliberations, the approximation-lattice ideas of limit and of continuity now come into their own. We won't dwell on this, but do point out one application. Let us say that a set-up is *finite* if it gives values other than ***None*** to only finitely many atomic formulas. Then every set-up s is the limit of a set of finite set-ups: $s = \bigsqcup X$ for some X a directed set of finite set-ups. This is important if the computer can directly represent only finite set-ups.

Moving up a level, we can also define a natural approximation-lattice ordering on the set ES of epistemic states. Naturally we want

$E \subseteq E'$ implies $E' \sqsubseteq E$,

since the smaller epistemic state E gives more definite information; but the converse won't do (unless both E and E′ are closed upward; see below). The right definition, yielding the above as a special case, is as follows:

$E \sqsubseteq E'$ iff every $s' \in E'$ is approximated by some $s \in E$.

For example, let

S	S′	S″
$s(p) =$ ***T***	$s'(p) =$ ***T***	$s''(p) =$ ***T***
$s(q) =$ ***None***	$s'(q) =$ ***T***	$s''(q) =$ ***None***
$s(r) =$ ***None***	$s'(r) =$ ***None***	$s''(r) =$ ***T***

Then E = {s} approximates E′ = {s′, s″}. Note that neither E nor E′ gives any information about q or r, but E′ tells us that $q \vee r$ is ***T***.

It is not true that this ordering of ES yields a lattice; antisymmetry fails. There are two ways to make it a lattice, both of which we mention and neither of which we employ. We begin the first by defining an equivalence relation by

E *is equivalent to* E′ iff each approximates the other.

We then "divide through" by this relation: take equivalence classes. This easily turns out to be a complete lattice, and a natural approximation lattice (partly since the equivalence is natural).

The second uses the method of "representatives" instead of equivalence classes. Define a state E as *closed upward* if $s \sqsubseteq s'$ and $s \in E$ imply $s' \in E$. Where CES is the set of all nonempty closed-upward states, it constitutes a natural approximation lattice ACES under the ordering defined above. Indeed in this case it is obvious that the ordering we defined above does in fact agree with the superset relation, so that obviously we have a complete lattice. One might worry, however, that we have cut out some interesting states. Not so: define the *upward closure of* E by:

C(E) = the family of set-ups approximated by some set-up in E.

Clearly C(E) is both upward closed and equivalent to E; so if we liked we could use C(E) as the "representative" of E. (Note also that E and E′ are equivalent just in case C(E) = C(E′); everything fits.)

But we choose to stay with ES and its ordering even though it is not a lattice, for, although both the lattice of equivalence classes and the lattice ACES are mathematically convenient (indeed we constantly rely on the convenience of the latter), they depart from practicality: the computer cannot work with the elements of these lattices since these elements are grossly infinite.

Let us then define AES as ES supplied with the ordering above and also with a couple of lattice-like operations which (1) give results equivalent to those obtained by passing through ACES and (2) *preserve finiteness*. The most natural meet operation is obviously just union:

$E \; E' = E \cup E'$

And the join:

$E \sqcup E' = \{s \cup s' : s \in E, s' \in E'\}$.

Also, analogously, for the general meet $\sqcap X$ and general join $\sqcup X$, where X is a subset of ES.

It is important that our valuation function E(A) is not only monotonic in the argument E but, in an appropriate sense, continuous in AES; in spite

of the fact that meets, used in the definition of E(A), are notoriously badly behaved in approximation lattices.

Certain elements of ES are of particular interest, namely those which characterize and are characterized by formulas. For each A, define the *truth set of A* and the *falsity set of A* as follows:

$$\text{Tset}(A) = \{\text{s: } \boldsymbol{T} \sqsubseteq \text{s}(A)\}$$
$$\text{Fset}(A) = \{\text{s: } \boldsymbol{F} \sqsubseteq \text{s}(A)\}$$

A is marked told True by all and only members of Tset(A), and told False by the members of Fset(A). Both of these sets are closed upwards, hence in CES. The next section in effect investigates some of their properties.

§81.3.3. Formulas as mappings: A new kind of meaning. Now we turn to a question of considerable interest, and a question on which our various approximation lattices can shed considerable light: How is the computer to interpret a truth-functional formula A as input? Clearly it is going to use A to modify its present epistemic state; and indeed it is not too much to say that defining how the computer uses the formula A to transform its present epistemic state into a new epistemic state is a way—and a *good* way—of giving A a meaning. Consequently we want to associate with formula A a transformation, a mapping from epistemic states into new epistemic states. Furthermore, we also want to know what the computer is to do when the formula A is *denied* to the computer; so actually we associate with a formula A *two* functions, one representing the transformation of epistemic state when A is affirmed, the other the transformation when A is denied. Let us call these two functions A^+ and A^-. How to define them?

Recall that A^+ is to map states into states: $A^+(\text{E}) = \text{E}'$. The key ideas in defining what we want E′ to be come from the approximation lattice. First, in our context we are assuming that the computer uses its input always to increase its information, or at least it never uses input to throw information away. (That would just be a different enterprise; it would be nice to know how to handle it in a theory, but we don't.) And we can say this accurately in the language of approximation: $\text{E} \sqsubseteq A^+(\text{E})$. Second, $A^+(\text{E})$ should certainly say no less than the affirmation of A: $\text{Tset(A)} \sqsubseteq A^+(\text{E})$. Third and lastly, we clearly want $A^+(\text{E})$ to be the *minimum mutilation* of E that renders A at least told True. "Minimum mutilation" is Quine's fine phrase, but in the approximation lattice we can give a sense to it that is *no longer merely metaphorical*: namely, we want the *least* of those epistemic states satisfying our first two conditions. That is, we should define

$$A^+(\text{E}) = \text{E} \sqcup \text{Tset}(A);$$

for that is precisely the minimum mutilation of E that makes A at least told True. (Recall that, in any lattice, $x \sqcup y$ is the "least (minimum) upper bound.")

Having agreed on this as the definition of A^+, it is easy to see that $A^-(\mathrm{E})$ should be the minimum mutilation of E that makes A at least told false:

$$A^-(\mathrm{E}) = \mathrm{E} \sqcup \mathrm{Fset}(A).$$

The above definitions accurately represent the meaning of A as input, but they do involve a drawback: the Tsets and Fsets may be infinite, or at least large, and so do not represent something the computer can really work with. For this reason, and also for its intrinsic interest, we offer another explication of A^+ and A^-, this time inductive, but still very much involving the idea of minimum mutilation.

First, what is the computer to do to its present epistemic state, when an atomic formula, p, is affirmed? Recalling that p must be marked at least told True in the result, and that it will not be such unless it is such in each member of E, it is clear that what the computer must do is run through each set-up in E and add told True to p. This will make p ***T*** if it was ***None*** before, it will leave it alone if it was already either ***T*** or ***Both***, and will make it ***Both*** if it was ***F***. And this is obviously the *minimum* thing the computer can do. Defining $\mathrm{p}_{\boldsymbol{T}}$ as that set-up in which p has ***T*** and all other atoms have ***None***, we can say this technically as follows (note where minimum mutilation comes in):

$$p^+\mathrm{E} = \{\mathrm{s} \sqcup p_{\boldsymbol{T}} : \mathrm{s} \in \mathrm{E}\}$$

And, with $p_{\boldsymbol{F}}$ defined similarly,

$$p^-\mathrm{E} = \{\mathrm{s} \sqcup p_{\boldsymbol{F}} : \mathrm{s} \in \mathrm{E}\}.$$

The union is in the approximation lattice AS of all set-ups.

The recursive clauses, which represent a way of giving meaning to the connectives (different from—though of course related to—the usual "truth conditions" account), now come easily.

$$(A\&B)^+ = A^+ \circ B^+$$

That is, to make $A\&B$ true by mimimum mutilation, first minimally mutilate to get B true, and then minimally mutilate the result to get A true as well. (The "$\circ$" is for composition of functions.) It had better turn out, and it does, that $(\mathrm{A\&B})^+ = (\mathrm{B\&A})^+$—i.e., that the order of minimal mutilation makes no difference. Next, obviously,

$$(\sim A)^+ = A^-.$$

And

$$(A \vee B)^+ = \lambda \mathrm{E}(A^+(\mathrm{E}) \sqcap B^+(\mathrm{E}))$$

That is, one makes the mimimum mutilations for A and B separately, and then one finds the best (maximum) among all the states that approximate both of these—which is just their set-theoretical union. For example, if E is a singleton $\{\mathrm{s}\}$ in which $p \vee q$ has ***None***, $(p \vee q)^+\mathrm{E}$ is obtained by "splitting" s

into two new states, in one of which p has $\boldsymbol{T}$ while q and everything else stay the same, and in the second of which q has $\boldsymbol{T}$ while p and everything else remain fixed.

We give the clauses for A^- without comment:

$$(\sim A)^- = A^+$$
$$(A\&B)^- = \lambda E(A^-(E) \sqcap B^-(E))$$
$$(A \vee B)^- = A^- \circ B^-$$

We have given two separate accounts of the meaning of A as input (affirmed or denied), so we had better observe that they agree. A third account of some merit begins by defining A^+ and A^- as functions from *set-ups* s into states E:

$$A^+s = \{s \sqcup s' \colon s' \in \mathrm{Tset}(A)\}$$
$$A^-s = \{s \sqcup s' \colon s' \in \mathrm{Fset}(A)\}$$

Then

$$A^+E = \cup\{A^+s \colon s \in E\}$$
$$A^-E = \cup\{A^-s \colon s \in E\}$$

And there are a number of other variations; e.g.,

$$(A\&B)^+E = A^+E \sqcup B^+E$$

§81.3.4. More observations. What we have done is use the approximation lattices not only to spell out in reasonably concrete terms what the computer is to do when it receives a formula as affirmed or denied, but, further, we have given a new theoretical account of the meaning of formulas as certain sorts of mappings from epistemic states into epistemic states. It is clear that there remains work to be done here in finding the right abstract characterizations and general principles, unless it has already been done somewhere or other; but we make a few comments.

To set the stage, recall that Scott has observed that the family of all *continuous* functions from an approximation lattice into itself (or indeed another) naturally forms a new approximation lattice, and it is important that our A^+ and A^- functions are members in good standing. But the A^+ functions (we may drop reference now to the A^- functions, since $A^- = (\sim A)^+$) form but a limited subset of all these functions, and it would be desirable to characterize an appropriate subset, without, however, leaning too heavily on linguistic considerations. One feature they all have in common is that they are one and all (weakly) *ampliative*:

$$E \sqsubseteq A^+E,$$

or, where I is the identity function on ES,

$$I \sqsubseteq A^+.$$

And this feature is a hallmark of our entire treatment: the computer is never to throw away information, only to soak it up. It is easy to see that the family of all continuous functions "above" $\boldsymbol{I}$ themselves form an approximation lattice—the lattice of all ampliative and continuous functions—which is closed under such pleasant operations as composition. ($\boldsymbol{I}$ is the bottom of this lattice.)

Another feature of the A^+ is that they are *permanent*: once A^+ is done to a state E, it stays done, and does not have to be done again, no matter how much the computer later learns. In symbols:

$$A^+\mathrm{E} \sqsubseteq \mathrm{E}' \text{ implies } A^+\mathrm{E}' = \mathrm{E}'.$$

These three features taken together can very likely be taken as a proper intrinsic characterization of the "kind" of functions represented by our truth-functional formulas. For a function f is continuous, ampliative, and permanent just in case f can be characterized as improving the situation by some fixed amount. That is, just in case there is some fixed element E_0 such that $\mathrm{f}(\mathrm{E}) = \mathrm{E} \sqcup \mathrm{E}_0$, for all E. And that sounds right.

The interested reader can verify that from these principles one can deduce what are perhaps the most amusing of the properties of the A^+ functions: composition is the same as join, hence is commutative and idempotent:

$$A^+ \circ B^+ = B^+ \circ A^+$$
$$A^+ \circ A^+ = A^+.$$

We do not like to leave this discussion on such an abstract note, and so we conclude with a more practical remark. What "permanence" in the above sense *means* for the computer is that it has a choice when it receives A as an input: it can, if it likes, "remember" the formula A in some convenient storage, or, if it prefers, it can "do" A to its epistemic state and then *forget* about it. Since the meaning of A is a permanent function, A will be permanently built into the computer's present *and future* epistemic states. In the next Part there will emerge important contrasts with this situation.

§81.3.5. Quantifiers again. Quantifiers introduce problems which *must* be worked through but which we do *not* work through. The chief difficulty comes from the fact that we must keep our set-ups and epistemic states finite for the sake of the computer, whereas a quantified statement contains infinitely much information if the domain is infinite. We are going to offer only some murky comments.

In the first place, we will stay with the substitutional interpretation of the quantifiers so as not to have to modify the definition of "set-up." So quantification is always with respect to a family of constants suitable for substitution: $\forall x Ax$ is the generalized conjunction of all its instances, and $\exists$xAx the generalized disjunction of its instances. So, given a substitutional range, the

reader can supply the right definitions for $s(\forall xAx)$ and $s(\exists xAx)$. Second, we are going to suppose that the substitution range is infinite; otherwise there is no problem. Third, with considerable hesitation, we are going to attach the substitution range to the entire epistemic state E, rather than permit the various set-ups in E to come with different substitutional ranges.

The problem is not really how to answer questions about quantified formulas (though there may be difficulty in practice), but in how to treat them as input. Perhaps it is obvious what we want for the existential quantifier: given $\exists xAx$ as input, add a *new* constant c to the substitutional range, and then make the minimum mutilation that makes Ac True. But we are not yet clear how to justify this procedure in approximation terms.

The universal quantifier as input is where the real problem lies: it can lead from a finite state E (i.e., a finite collection of finite set-ups) to an infinite state E′. What is probably best is to apply the universal quantifier (mutilate minimally to make an instance true) only for a while; this will force the computer to *remember* the universally quantified formula so that it can be applied again later, if necessary. (What counts as "necessary" is: as much as is needed to answer the questions asked.) The various finite states obtained by repeatedly applying $\forall xAx$ in this way clearly have *as a limit* the minimum mutilation in which $\forall xAx$ is True.

Some of what is needed can be better appreciated from the point of view of Part 3, and we drop the matter for now.

§81.4. Part 3. Implicational inputs and rules. In Part 1 we pretended that all information fed into the computer was atomic, so we could get along with set-ups. In Part 2 we generalized to allow information in the form of more complex truth-functional formulas, a generalization which required moving to epistemic states. Now we must recognize that it is *practically* important that sometimes we give information to the computer in the form of rules that allow it to modify its own representation of its epistemic state in directions we want. In other words, we want to be able to instruct the computer to make inferential moves that are not mere tautological entailments. For example, instead of physically handing the computer the whole list of Series winners and nonwinners for 1971, it is obviously cheaper to tell the computer: "the Pirates won; and further, if you've got a winner and a team not identical to it, that team must be a nonwinner" (i.e., $\forall x\forall y(Wx\&x \neq y \rightarrow \sim Wy)$). In the presence of an obviously needed table for identity and distinctness or else in the presence of a convention that different names denote different entities (not a bad convention for *practical* use in many a computer setting), one could then *infer* that "The Orioles won" is to be marked told False.

Your first thought might be that you could get the effect of "given A and B, infer C," or "if A and B, then C," by feeding the computer "$\sim A \vee \sim B \vee C$." But that won't work: the latter formula will tend to *split* the set-up you've

got into three, in one of which A is marked told False, etc.; whereas what is wanted is (roughly) just to improve the single set-up you've got by adding told True to C provided that A and B are marked told True (and otherwise to leave things alone). (Connections of this idea with that of Belnap 1970 and 1973 need exploring.) It is (roughly) this idea we want to catch.

§81.4.1. Implicational inputs. Let us introduce "$A \rightarrow B$" as representing the implication of A to B; so what we have is notation in search of a meaning. But we have found in the previous section just the right way of giving meaning to an expression construed as an input: the computer is to improve its epistemic state in the minimum possible way so as to make the expression True. So let us look forward to treating $A \rightarrow B$ as signifying some mapping from states into states such that $A \rightarrow B$ is True in the resultant state.

Obviously if we are to pursue this line, we must know what it is for $A \rightarrow B$ to be True in a state. This is a delicate matter. One definition that suggests itself is making $A \rightarrow B$ True in a state E just in case $\mathrm{E}(A) \leq \mathrm{E}(B)$ (in the logical lattice **L4**); but although we don't have any knock-down arguments against the fruitfulness of this definition, we are pretty sure it is wrong. We think it will be more fruitful to define $A \rightarrow B$ closer to the following: modify *every* set-up you are considering to make $A \rightarrow B$ True in it. So let us first define what it is for $A \rightarrow B$ to be "True in a set-up"; and naturally for this we turn to the logical lattice **L4**; let us specify that $A \rightarrow B$ is True in s just in case $\mathrm{s}(A) \leq \mathrm{s}(B)$ (Note that we do *not* give $A \rightarrow B$ values in **4**; $A \rightarrow B$ is just True or False in s, never both or neither, since not merely "told.")

It might be tempting now to define $A \rightarrow B$ as True in state E if True in every set-up s in E, and False otherwise, but that would be wrong. The reason is that the Truth of $A \rightarrow B$ is *not* closed upward; $\mathrm{s} \sqsubseteq \mathrm{s}'$ and $A \rightarrow B$ True in s do not together guarantee that $A \rightarrow B$ is True in s'. But epistemic states are supposed to be equivalent to their upward closures. The next thing to try is to look just at the *minimal members* M(E) of each state E, i.e., those set-ups in E which are minimal with respect to the approximation ordering between set-ups. For, in any state E in which every set-up is approximated by some minimal set-up, nonminimal set-ups (those not in M(E)) can be thought of as redundant. In particular they do not contribute to the value of any formula and should not contribute to the value of implications. So it would be plausible to define $A \rightarrow B$ as True in a state if it is True in every minimal member. And indeed this will work if E is finite or if every s in E is finite; for then every descending sequence

$$\mathrm{s}_1 \sqsupseteq \ldots \mathrm{s}_i \sqsupseteq \ldots$$

in E is finite, and so in fact M(E) is equivalent to E. Of course for real applications on the computer this will always be so. But let us nevertheless give a definition that will work in the more general case: $A \rightarrow B$ is True in

E if, for every s∈E, there is some s′∈E such that s′ ⊑ s, and $A \to B$ is True in s′.

We claim for this definition the merit of passing over equivalent states: given E and E′ equivalent, $A \to B$ will have the same truth value in each. The reason that if $A \to B$ is True in the closure of E then it is True in E as well (the hard part) is that there cannot be in the closure of E an infinitely descending chain of set-ups in which the truth value of $A \to B$ changes infinitely often. Sooner or later as you pass down the chain, the truth-value of $A \to B$ will have to stabilize as either True or False. And under the hypothesis that $A \to B$ is True in the closure of E, in each chain it will have to stabilize as True; which is enough to get it True in E. And the reason that there cannot be an infinitely descending chain of set-ups in which the truth value of $A \to B$ flickers is that any such flicker must be caused by a change in either s(A) or s(B); but since this function is monotonic in s, once having changed the value of A or B in the only permitted (downward) direction, one can never change it back up again. So at most the value of A can change twice, and similarly for B; which means that $A \to B$ can change at most four times.

One more note of profound caution: the notion of the Truth of $A \to B$ in E is dramatically different from the notion of A's being told True in E, in that the former is *not* monotonic in E, whereas the latter is: E ⊑ E′ guarantees that if A is at least ***T*** in E then it is so in E′, but does not guarantee that if $A \to B$ is True in E it is so in E′. (The falsity of $A \to B$ fares no better.) We shall see later how this influences the computer to manipulate $A \to B$ and A quite differently; now, however, we remark that this fact is not in conflict with Scott's thesis, since we have not got something that can be represented as a function from one approximation lattice into another. In particular, the usual characteristic function representing the set of E in which $A \to B$ is True will not work, since the two truth values True and False do not constitute an approximation lattice.

Now back to our enterprise of defining $A \to B$ in such a way as to make it a mapping from epistemic states E to states E′ in such a way as to represent minimum mutilation yielding Truth—in exact analogy with our results in the previous section.

Since we know what it is for $A \to B$ to be true in s, we know that it has a truth set: Tset $(A \to B) = \{s: A \to B \text{ true in } s\}$. So one might try just defining

$$(A \to B)^+ E = E \sqcup \text{Tset}(A \to B)$$

as before. There may be something in the vicinity that works, but this doesn't; since one of the set-ups in which $A \to B$ is True is that in which every atomic formula has ***None***, this $(A \to B)^+$ is just the identity function (up to equivalence). It is also worth noting that Tset$(A \to B)$ is not closed upward and so not well-behaved. Nor will it do to try to close it upward—by the remark above, that would yield the family of *all* set-ups. In any event, we take a

different—and, we think, intuitively plausible—path to defining $(A \rightarrow B)^+$ as a function that minimally mutilates E to make $A \rightarrow B$ true.

We propose first to define $A \rightarrow B$ on set-ups s, looking forward to the following extension to states:

$$(A \rightarrow B)^+ \mathrm{E} = \cup \{(A \rightarrow B)^+ \mathrm{s}: \mathrm{s} \in \mathrm{E}\}$$

So we are up to defining $(A \rightarrow B)^+$ on a set-up s—with the presumption doubtless that the value will be some state E′ (we may have to "split" s). The idea is, as always, that we want to increase the information in s as little as possible so as to make $A \rightarrow B$ true. If we keep firmly in mind that "increase of information" is no mere metaphor, but is relative to an approximation lattice, it turns out that we are guided as if by the hand of the Great Logician.

One case is easy. If $A \rightarrow B$ is *already* True in s, the minimum thing to do is to just leave s alone. Now in order to motivate the definition to come, consider all the ways that $p \rightarrow q$, for example, could be *False* in s. The possibilities, which refer to the *logical* lattice **L4**, are laid out under p and q below (ignore for now the right-hand column).

p	q	Essential to make $p \rightarrow q$ True
T	***None***	Raise q to ***T***
T	***Both***	Raise p to ***Both***
T	***F***	Raise p to ***Both*** and q to ***Both***
None	***F***	Raise p to ***F***
Both	***F***	Raise q to ***Both***

Now try the following. Keep one eye on the logical lattice **L4** (§81.2.2) and the other on the approximation lattice **A4** (§81.2.1) and use your third to verify the claims made in the right-hand column. For example, the first entry says, in effect, that if $p \rightarrow q$ is False because p is ***T*** and q is ***None*** then *it* does no good to raise p (in the approximation lattice **A4**), for the only place to which to raise it is to ***Both***; and (in the logical lattice **L4**) that still doesn't imply q (make $p \rightarrow q$ True). So q must be raised. (An important presupposition of these remarks is that we may speak only of "raising" in the approximation lattice **A4**, never of "lowering"; the computer is never to treat an input as *reducing* its information, never to treat it as a cause to "forget" something. And you will recall that this constraint is a local one, certainly not part of what we think is essential to the Complete Reasoner.)

Next note the following analysis of the table, where s is the current set-up and E′ is the new state. All the raisings of q occur when $\boldsymbol{T} \sqsubseteq \mathrm{s}(p)$, and all

the raisings of p occur when $F \sqsubseteq s(q)$. Further, the raising of q consists in always making $T \sqsubseteq E'(q)$ and the raising of p consists in making $F \sqsubseteq E'(p)$. That is, as might have been expected, making $p \to q$ True consists in making q have at least T when p does and in making p have at least F when q does.

Let us divide the problem (and abandon the special case of atomic formulas). One thing we must do is make B have at least T when A does. Let us call the corresponding statement: $A \to_T B$. We want to make B told True if A is, and in a minimal way. But we *already* know the minimal way of making B told True. So the following definition of $(A \to_T B)^+$ is pretty well forced:

$$\begin{aligned} (A \to_T B)^+ s &= B^+\{s\} && \text{if } s \in \mathrm{Tset}(A); \text{ i.e., if } T \sqsubseteq s(A), \\ &= \{s\} && \text{if } s \notin \mathrm{Tset}(A); \text{ i.e., if } T \not\sqsubseteq s(A). \end{aligned}$$

This account of $(A \to_T B)^+$ matches very well the intuitions that led Ryle 1949 (see §6) to say that "if-then"s are inference tickets. For $(A \to_T B)^+$ is exactly a license to the computer to infer the conclusion whenever it has got the premiss in hand. For example, if it finds that "The Pirates won" is marked T, then "The Pirates won $\to_T$ the Orioles didn't" will direct it to make the minimum mutilation that marks "the Orioles didn't" with at least T. (Recall from the previous section that B^+ is the minimum mutilation making B at least T.)

There is already much food for thought here, and a host of unanswered questions. We do note that Scott's thesis is not violated: $(A \to_T B)^+$ is indeed a continuous function from the space of set-ups to that of states—and, with the previous extension, from states into states. That it is depends on the fact that Tsets are (1) always closed upward and (2) "open": if $\bigsqcup X \in \mathrm{Tset}(A)$ for directed X, then $x \in \mathrm{Tset}(A)$ for some $x \in X$. (The topological language fits the situation: it means that no point in X can be approached as the limit of a family of points lying entirely outside of X.) The point of this remark is to draw the consequence that we cannot sensibly use $A \to_T B$ in the absence of these conditions; hence, since the Tset for $A \to_T B$ is *not* closed upward, we cannot make sense of $(A \to_T B) \to_T C$. In contrast, all we need from B is the continuity of B^+; so $A \to_T (B \to_T C)$ is acceptable. (Note how the approximation idea and Scott's thesis guide us through the thicket.)

For its intrinsic interest, note that, in the lattice of all ampliative functions, we have

$$((A \to_T B)^+ \circ A^+) \sqsupseteq B^+$$

but not

$$(A^+ \circ (A \to_T B)^+) \sqsupseteq B^+.$$

Maybe this has something to do with some of the nonpermutative logics, and maybe not.

Turning back now to our principle task, the defining of $(A\rightarrow B)^+$, we have completed part of our task by defining $(A\rightarrow_{\boldsymbol{T}} B)^+$, which makes B true if A is. The other part is by way of the function

$$\begin{aligned}(A\rightarrow_{\boldsymbol{F}} B)^+\mathrm{s} &= A^-\{\mathrm{s}\} && \text{if } \mathrm{s}\in \mathrm{Fset}(B);\ \text{i.e., if } \boldsymbol{F}\sqsubseteq \mathrm{s}(B),\\ &= \{\mathrm{s}\} && \text{if } \{\mathrm{s}\}\notin \mathrm{Fset}(B),\ \text{i.e., if } \boldsymbol{F}\not\sqsubseteq \mathrm{s}(B).\end{aligned}$$

This is the function that makes A told False, minimally, if B is.

Before pushing on to define $(A\rightarrow B)^+$, let us pause to note just a thing or two about $(A\rightarrow_{\boldsymbol{T}} B)^+$. This family of functions has in common with the A^+ that each is ampliative:

$$\boldsymbol{I}\sqsubseteq (A\rightarrow_{\boldsymbol{T}} B);\ \text{i.e., } \mathrm{E}\sqsubseteq (A\rightarrow_{\boldsymbol{T}} B)^+\mathrm{E}.$$

In contrast, however, these new functions are *not* "permanent" in the sense defined at the end of Part 2. That means that, once the computer has "done" $(A\rightarrow_{\boldsymbol{T}} B)^+$, it may have to do it again; this is a consequence of the fact that the truth set of $A\rightarrow B$ is not upward closed: adding new information can falsify $A\rightarrow B$. But there is one property in the vicinity that $(A\rightarrow_{\boldsymbol{T}} B)^+$ shares with A^+: at least one doesn't have to do it twice in a row:

$$\mathrm{f}\circ\mathrm{f} = \mathrm{f}$$

for $\mathrm{f} = (A\rightarrow_{\boldsymbol{T}} B)^+$.

Closely related to the permanence-impermanence distinction between the two sorts of ampliative functions is the way they behave under composition: all the truth-functional ampliative functions permute with each other $(A^+\circ B^+ = B^+\circ A^+)$, but the $\rightarrow_{\boldsymbol{T}}$ functions permute neither with each other nor with the truth functions. The clearest example of the latter is the inequality:

$$(p^+\circ(p\rightarrow_{\boldsymbol{T}} q)^+)\neq((p\rightarrow_{\boldsymbol{T}} q)^+\circ p^+)$$

Applying the right-hand side to an s in which p and q each have ***None*** yields a state in which first p is made told True, and then, as a consequence of this, q is made told True, too. But applying the left-hand side to s does not fare so well: $p\rightarrow_{\boldsymbol{T}} q$ does no work, since p is not at least $\boldsymbol{T}$ in s; so the outcome is only the marking of p as told True *without* changing q.

By noting that $(A\rightarrow_{\boldsymbol{F}} B)^+ = (\sim B\rightarrow_{\boldsymbol{T}}\sim A)^+$, we can be sure that this function has both the virtues and the shortcomings of $(A\rightarrow_{\boldsymbol{T}} B)^+$—except that it has the additional shortcoming that not only is $(A\rightarrow_{\boldsymbol{F}} B)\rightarrow_{\boldsymbol{F}} C$ impossible, since the falsity set of $(A\rightarrow_{\boldsymbol{F}} B)^-$ is not closed upward, but so is $A\rightarrow_{\boldsymbol{F}}(B\rightarrow_{\boldsymbol{F}} C)$, since $(B\rightarrow_{\boldsymbol{F}} C)^-$ is not defined. (We can if we like have $A\rightarrow_{\boldsymbol{T}}(B\rightarrow_{\boldsymbol{F}} C)$.)

The shortcomings of the arrow functions make us see that we cannot define $(A\rightarrow B)^+$ as simply the composition of $(A\rightarrow_{\boldsymbol{T}} B)^+$ and $(A\rightarrow_{\boldsymbol{F}} B)^+$. For

$A \to B$ might not be True in the result. Intuitively, $(A \to_F B)^+$ might cause nothing to happen, since B is ***None*** in the set-up s in question; while $(A \to_T B)^+$ causes B to be marked not only told True (since A is) but told False *as well*. This can happen if B is a formula like $p\&\sim p$ which cannot be made told True without being made told False *as well*. Then, if A still has the value ***T***, $A \to B$ will be false. So the composition of $(A \to_T B)^+$ with $(A \to_F B)^+$ (in either order) is *not* the minimum increase making $A \to B$ true in the result. As a special solution to this problem; one finds that $(A \to_F B)^+ \circ (A \to_T B)^+ \circ (A \to_F B)^+$ works admirably: first make A False if B is; then make B True if A is; then; once more, make A False if B is. Since $A \to B$ *is* true in the result, one need do nothing else; one has indeed found the minimum. In particular,

$$((A \to B)^+ \circ (A \to B)^+)\mathrm{s} = (A \to B)^+ \mathrm{s}$$
$$(A \to B)^+ = (A \to_T B)^+ \circ (A \to_F B)^+ \circ (A \to_T B)^+.$$

So we take this as a definition of what $A \to B$ *means* as a mapping of epistemic states into epistemic states.

We conclude this section with two remarks. First, we have offered no *logic* for rules $(A \to B)^+$; there is just much work to be done.

Second, $A \to B$ has been construed as a rule and has been given "input" meaning. It has been given no output meaning, and it is not intended that the computer answer questions about it. In particular, we have given no meaning to *denying* $A \to B$; $(A \to B)^-$ has not been given a sense. We are not sure whether this is a limitation to be overcome or just a consequence of our presenting $A \to B$ as a rule; for we do not know what it would mean to tell the computer *not* to use the rule $(A \to B)^+$. One might try to give sense to $(A \to B)^-$ by instructing the computer to make $\mathrm{E}(A) \not\subseteq \mathrm{E}(B)$; but this is an instruction that it is not always possible for the computer to carry out. Or the counterexample idea of §49 might work.

§81.4.2. Rules and information states. This last subsection is going to be altogether tentative, and altogether abstract, with just one concrete thought that needs remembering, which we learned from Isner: probably the best way to handle sophisticated information states in a computer is by a judicious *combination* of tables (like our epistemic states) and rules (like our $A \to B$ or a truth-functional formula that the computer prefers to remember, or a quantificational formula that it must remember). For this reason, as well as for the quite different reason that some rules may have to be used again (are not permanent, must be remembered), we can no longer be satisfied to represent what the computer knows by means of an epistemic state. Rather, this must be represented by a *pair* consisting of an epistemic state *and* a set of rules:

$\langle \mathrm{R}, \mathrm{E} \rangle$.

E is supposed to represent what the computer explicitly knows, and is subject to increase by application of the rules in the set R. For many purposes we should suppose that E is finite, but for some not.

Let us dub such a pair an *information state*, just so that we don't have to retract our previous definition of "epistemic state." But what is a rule? Of what is R a set? A good thing to mean by *rule*, or *ampliative* rule, in this context, might be: *any* continuous and ampliative mapping from epistemic states into epistemic states. As we mentioned above, the set of *all* continuous functions from an approximation lattice into itself has been studied by Scott; it forms itself a natural approximation lattice. It is, furthermore, easy to see that the ampliative continuous functions form a natural approximation lattice, and one which is an almost complete sublattice of the space of all the continuous functions: all meets and joins agree, except that the join of the empty set is the identity function I instead of the totally undefined function. Intuitively: the effect of an empty set of rules is to leave the epistemic state the way it was.

So much for the general concept of rule, of which the various functions A^+, A^-, $(A\rightarrow_T B)^+$, $(A\rightarrow_F B)^+$, and $(A\rightarrow B)^+$ are all special cases. We now have to say what a set R of rules *means*. Of course we want to express it as a mapping from epistemic states into epistemic states. Let us begin by saying that a rule ρ is *satisfied in* a state E if applying it to E does not increase information: $\rho(E) = E$, and by saying also that a set R of rules *is satisfied in* E if all its members are. Then what we want a set of rules to do is to make the minimum mutilation of E that will render all its members satisfied. Even if R is a unit set, simple application of its member might not work to satisfy it. And even if R is a finite set of rules, each of which is satisfied after its own application, the simple composition of R might not be adequate: All of this can be derived from considerations we adduced in defining $(A\rightarrow B)^+$. But there is a general construction which is bound to work.

Let R be a set of rules. Let R° be the closure of R under composition. This is a directed set; the composition $f\circ g$ of f and g will always provide an upper bound for both f and g if they are monotonic and ampliative. Now take the limit: $\bigsqcup R^\circ$.

Claim: take any E and any set R of rules. Then $\bigsqcup R^\circ(E)$ is the minimum mutilation of E in which all rules in the set R are satisfied. (Below we write R(E) for $\bigsqcup R^\circ(E)$.)

In this way we give meaning to the pair consisting of an epistemic state E and a set of rules R. There is that state R(E) consisting of "doing" the rules in all possible ways to E, and it is in regard to this state that we want our questions answered in the presence of E and R. Of course R(E) can be infinitely far off from E. This will certainly happen if the computer is dealing

with infinitely many distinct objects and some rule involves universal quantification; so in practice, "I don't know" might have to mean either: "I haven't computed long enough" or "I have positive evidence that I haven't been told."

Because of the importance of computers that maintain both (1) sets of rules and (2) tables (epistemic states), the idea of information states $\langle R, E\rangle$ should be studied in detail. We close this section with just a few idle definitions in the area which might or might not turn out to be fruitful.

When are two states equivalent? There seem to be at least two ideas: $\langle R_1, E_1\rangle$ is *currently equivalent* to $\langle R_2, E_2\rangle$ just in case $R_1(E_1) = R_2(E_2)$; which is to say, they give the same answers to the same questions. And they are *strongly equivalent* if adding the same information to each always produces currently equivalent results: $\langle R_1, E_1 \sqcup E\rangle$ is currently equivalent to $\langle R_2, E_2 \sqcup E\rangle$, for all E. Such information states would answer alike not only all present questions, but also all future questions asked after the addition of the same information to each.

We defined a rule ρ as *satisfied* in an epistemic state E if $\rho(E) = E$. We could similarly define a rule as satisfied in an information state $\langle R, E\rangle$ in one of two ways: *currently satisfied* if satisfied in E, and *ultimately satisfied* if satisfied in R(E). A third notion brings in just the set R: perhaps saying that a rule ρ is *in force* in R might be defined by: $\rho \sqsubseteq R$; i.e., ρ approximates R. But this is *not* a "relevant" idea of being in force; e.g., for each A the rule $(A \to A)^+$ is in force in every R.

PROBLEM. What is a relevant idea?

§81.4.3. Closure. Lest it have been lost, let us restate the principal aim of this section: to propose the *usefulness* of the scheme of tautological entailments as a guide to inference in a certain setting, namely, that of a reasoning, question-answering computer threatened with contradictory information. No reader of this book can possibly suppose that Larger Applications have not occurred to us; e.g., application of some of the ideas to a logic of imperatives, or to doxastic logic, or to the development of The One True Logic. But because of our fundamental conviction that logic is occasionally practical, we did not want these possibilities to loom so large as to shut out the light required for dispassionate consideration of our far more modest proposal.

§82. Rescher's hypothetical reasoning: An amended amendment. Rescher 1964—henceforth HR—proposes a way of reasoning from a set of hypotheses that may include some of our beliefs and also hypotheses contradicting those beliefs. The aim of this section is to point out what we take to be a fault in Rescher's proposal and to suggest a modification of it, using a nonclassical logic, which avoids that fault. We neither attack nor defend the broader