

POINTERS TO PROPOSITIONS*

HAIM GAIFMAN

1. INTRODUCTION

The semantic paradoxes, whose paradigm is the Liar, played a crucial role at a crucial juncture in the development of modern logic. In his 1908 seminal paper, Russell outlined a system, soon to become that of the *Principia Mathematicae*, whose main goal was the solution of the logical paradoxes, both semantic and set-theoretic. Russell did not distinguish between the two and his theory of types was designed to solve both kinds in the same uniform way. Set theoreticians, however, were content to treat only the set-theoretic paradoxes, putting aside the semantic ones as a non-mathematical concern. This separation was explicitly proposed, eighteen years after Russell's paper, by Ramsey, though he, like Russell, advocated a system that addresses both kinds. Since then, the semantic paradoxes have been viewed within the perspective of the theory of truth, where they have occupied a respectable niche, but one of rather specialized interest.

In this work I shall try to move the issues arising from the semantic paradoxes to a more central place within the philosophy of language. It is not so much the paradoxes themselves as what they reveal about mechanisms incorporated in natural language that is philosophically so significant. These are mechanisms that enable us to make within the same language statements that, in the usual order of things, would require semantic ascents to metalanguages. The implications of the emerging picture extend well beyond the specific concerns of the paradoxes.

The work has philosophical and technical aspects. I shall try to make the philosophical points clear, without relying too much on the more technical parts that occupy most of section 4 and part of section 5. The emerging picture is roughly this. Statements are made, or can be made, by means of objects—the “pointers” of the title—which form a kind of network. The semantics consists of recursive rules, by which every pointer is either given a truth-value—signifying the truth or the falsity of the statement expressed through it—or is classified as a failure, i.e., as failing to express something true or false. Each pointer has an associated sentence (the sentence it “points” to), which, in turn, may refer, either by name or through quantification, to pointers. By using a pointer, one can therefore make, or try to make, a statement about statements. These cross references generate the network. In general, the associated sentence is not sufficient for determining the statement made through a pointer. The pointer's place in the network enters as well. *Modo grosso* the place determines the metalevel at which the sentence is read; that is,

*I would like to thank my colleagues Isaac Levi and Rohit Parikh and my students Frederico Marulanda and Gurpreet Rattan for useful comments on an earlier draft of this work.

the metalanguage to which the sentence would belong—had we operated within a hierarchy of languages instead of using pointers. Thus, through pointers, we can express in a single language statements that would otherwise require metalinguistic ascent. This, I shall argue, happens in natural language.

The semantics is, to a large extent, holistic. Although it has a compositional component, which enters through the associated sentences, it is on the whole non-compositional: what a pointer expresses is not derived only from its associated structure and the structure's parts, but also from contextual parameters of the global network.

Usually, contextual aspects are handled by appeal to informal pragmatics. Here, by contrast, intricate contextual factors are given a formal rigorous modeling. The thought arises that the difference between pragmatics and semantics is not as sharp as it might seem. The first can become the second, upon rigorous systematization. Another point of interest is the non-compositionality of the semantics. Non-compositional semantics, sometimes described as operational, are well-known in the theory of programming languages. But the incorporation of such a semantics into a theory of truth is uncommon in the philosophy of language.

In the formal system pointers appear as “technical” elements, subject to well-defined rules. But since they determine how sentences are read, they should be seen in a wider perspective as representing points of view, stances, ways of interpreting given assertions. Such a perspective is present even in simple illustrative examples. It becomes clearer in section 5, where distinctions between various kinds of pointers serve as a basis for a ranking that enters into the semantic rules.

An earlier, different system was the subject of “Pointers to Truth” [Gaifman 1992], abbreviated henceforth as PTT. Developing the earlier ideas, I was led to a different and simpler system of rules, which results in a considerably simpler setup. The required structural elements have been reduced to a bare minimum, which makes it possible to apply the method to a wider variety of pointer systems. Various philosophical aspects have gained thereby in clarity.

This paper is self-contained. Knowledge of PTT is not presupposed. The basic example of PTT, the so called *two-line puzzle*, will serve also here as a standard illustration and a launching point. After the first moves, the rest of this introduction is devoted to elaborations of the remarks made above. Section 2 contains further arguments, intended to clarify and establish the correctness of the proposed analysis. Section 3, which is more or less self-contained, contains an outline of the system; pointers are introduced and discussed and the concept of a proposition that figures in this work is clarified; also discussed are some other issues, including the Strengthened Liar. Section 4 contains the more technical part of the work, where the basic formal system is fully presented. I have avoided going into proofs, when these are too long; but some of the shorter proofs are included. Section 5 introduces a variant based on a ranking of pointers, which enhances the class of expressible propositions. It has technical, as well as philosophical sides.

For didactic purposes I have postponed the introduction of pointers to section 3. Sentence-tokens, which are the standard examples of pointers, will serve until then.

Tokens of sentences are physical objects, organized either spatially (inscriptions) or temporally (utterances), which display abstract sentences: the tokens' *types*. It is a commonplace that, as a rule, truth and falsity depend on tokens, over and above their types. 'I am hungry' has different truth-values, depending on who makes the utterance. The same goes for all cases that involve indexicality, either explicitly through words such as 'I', 'you', 'here', etc., or implicitly—by contextual factors surrounding the utterance or the inscription.

An altogether different kind of token-dependence is involved in the use of semantic predicates, 'true', 'false', 'necessary' (when this is construed as a predicate over sentences) and others. While indexicality works in familiar ways, the token dependence of the second kind is a deep phenomenon, whose very existence, not to speak of its underlying mechanism, is far from clear. The following two-line puzzle will serve as our standard example.

Line 1 The sentence on line 1 is not true.

Line 2 The sentence on line 1 is not true.

By a well-worn argument the sentence on line 1 is not true (if the sentence is true, then it is not true). Writing this conclusion on line 2, one finds that one has repeated the very same sentence. If the sentence on line 1 stands condemned so does the sentence on line 2. Yet the latter expresses a true conclusion. Note that, in order to state this conclusion, one cannot but repeat the sentence (as I have just done: "By a well-worn argument the sentence on line 1 is not true.") or use some equivalent phrasing in which the sentence on line 1 is referred to by a different name. Hence, I argued in PTT, the two tokens mean different things. The first is not true; the second states truly this very same fact.

Let us take a closer look at the failure of the line 1 sentence. The standard evaluation rule for a sentence of the form 'The sentence written in/on ... is true' is roughly this:

- (*) *Go to ... and evaluate the sentence written there. If that sentence is true, so is 'The sentence written in ... is true', else the latter is false.*

To get the truth-value of the negated sentence ('The sentence written in/on ... is not true') we should apply (*) and follow it up by applying the rule for negation (where the latter step is supposed to reverse the truth-value). In the case of the line 1 sentence, the evaluation does not terminate; the sentence sends us back to the starting point. Thus, we get a *closed loop*. The "go-to" command makes the referring of 'The sentence written in/on ...' operationally explicit. But the loop is not due merely to self-reference. Crucially, the instruction tells us to evaluate the sentence for truth-value, thereby directing us back to (*). Had the instruction been non-semantic, say to count the number of words in the sentence ('The sentence on line 1 has an odd number of words') or to perform an orthographic check ('The sentence on line 1 contains no misspellings'), there would have been no loop and no paradox.

The closed loop yields a non-terminating evaluation, and for this reason alone the sentence is not true. It also makes possible the contradiction ensuing from the

assumption that the sentence is true; but it does not imply such a contradiction in general. Had the sentence been ‘The sentence on line 1 is true’, we would have gotten the Truth Teller (which attributes truth to itself and says nothing besides). Like the Liar, it is not true by virtue of its non-terminating evaluation. But the truth of the Truth Teller does not lead to contradiction, and neither does its falsity. We shall later see how the distinction between the Liar and the Truth Teller is expressed in the proposed system.¹ The conclusion that the line 1 sentence is not true reflects the realization that the straightforward implementation of (*) fails. It is expressed by using tokens different from the line 1 token, e.g., the other tokens on this page, including the one on line 2. The other tokens succeed because they are external to the loop produced by the first token. We can already see how different tokens mark different levels. The first token is *in* the loop, the second is, in a sense, *about* it. More of this in section 2.

One infers that, in this and in similar situations, truth-values should be assigned not to sentence-types but to their tokens, and that the evaluation rule should be modified so as to make the token on line 2 true. If this analysis—the argument for which I shall elaborate later—is correct, we are faced with the problem of finding a prescription for assigning truth-values to tokens in general. Loops can arise from indirect self-reference, which involves many sentences in diverse complicated ways. There might be also other phenomena that would cause sentence-tokens to fail. Our definition should decide when tokens fail and when this failure is expressed by other tokens. Like the language used in the two line puzzle, our language should include its own truth predicate. But since truth is now assigned to tokens, the predicate should be over tokens rather than types.

The system outlined in PTT (and, more so, the one presented here) meets this challenge. It applies to a full-fledged first-order language that has names referring to, and variables ranging over its sentence-tokens, and semantic predicates— $Tr()$ for ‘true’ and $Fa()$ for ‘false’—taking token-names as arguments. There should be nothing mysterious about predicating truth and falsity of tokens. A token, p , is true if it succeeds in expressing something true; is false, if it succeeds in expressing something false; is neither true nor false (stated as $\neg Tr(p) \wedge \neg Fa(p)$), if it fails to express something that has a truth-value.

Sentences such as ‘What is written in ... is false’, ‘Whatever Jane said is true’, are easily recast as sentences of the formal language. The setup is sufficiently general so as to make room for arbitrary networks generated by tokens that refer to tokens. The evaluation procedure (the formal method of assigning truth-values) accords, in the case of the two-line puzzle, with the above analysis and yields the desired verdict. The same works for a wide variety of indirect self-referential cases, including, roughly speaking, all those in which the self-reference involves essentially a finite number of tokens. Certain intricate, infinitary cases fall outside. In those cases—some of which will be discussed in the last two sections—there are no tokens that express the desired propositions and we are forced to ascend to a metalanguage. Other variants of the system are designed to take care of some of these examples.

The proposed system introduces an essential non-compositional element into the semantics: the meaning of a sentence-token is determined not only by the structure of the sentence (as a type) and by the meanings its components, but also by the token's place within a global network of mutually referring tokens. Contextual dependencies are of course common in the cases of indexicals and demonstratives. But these are determined by rules relating directly to human intentional acts, rules whose analysis—since they lack an interesting recursive or combinatorial structure—is not profitably pursued by formal modeling. The mechanism of indexicals and demonstratives can be separated from the standard compositional semantics and their contributions to the meaning of larger units can be set aside, as it is done by Kaplan [1989], under the name *character*.

The contextual dependency that enters through the use of semantic predicates is a different matter altogether.² An intricate recursive procedure is at work, which, by defining a truth-value assignment, determines what tokens say. Roughly, it determines the metalinguistic level at which the tokenized sentence would be interpreted—had we used a type-based semantics.

The truth-value assignment is defined by clauses, of which some are compositional and some contextual. But since both kinds are interleaved when the recursion is carried out, there is no separation of the procedure into compositional and contextual parts. We can distinguish, in the evaluating process, compositional segments from contextual ones, yet each segment relies, so to speak, on the segments that precede it.

Now the role of formal modeling in semantic theory should be properly understood. The formalism does not imply an agenda for formalizing natural language, but is rather a tool: a formal—hence artificial—yard stick that serves to bring forth fundamental patterns of conceptual thought. It is not implied that people are aware of and follow the turns and twists of the evaluation procedure, just as it is not implied that they are aware of and follow the intricacies of formal quantification. Neither does the viewing of language through a system that makes no explicit reference to speakers' intentions and acts indicate that the latter are ignored. It is only assumed that what the speakers say can be fruitfully analysed, by using a formal system as a match, without introducing explicitly the speakers' intentions. This methodology underlies the general mainstream project of logic. Some further comments on semantic theory will help to put the implication of the proposed system into a wider perspective.

When philosophers of language speak of a semantic theory, usually they have in mind what is known as *compositional semantics*: a systematic account of sentences and other syntactic units as structured objects, and of the way in which a sentence's truth-value derives from semantic features of its components.³ These features are often represented as *semantic values*: semantic entities associated with syntactic constructs. The semantic value of a phrase is then uniquely determined by its structure and the semantic values of its components. The semantic value of a sentence either determines its truth-value, or is simply identified with it. A scheme of this kind, traceable to Frege, applies to Tarski's semantics for formal languages.

Less well-known among philosophers is the generalization of that scheme and the rigorous form given to it in the analysis of programming languages ([Scott 1970], [Scott and Strachey 1971]), where it came to be known as denotational semantics (cf. [Gaifman and Shapiro 1989] for a succinct formal description; for a detailed discussion see [Winskel 1993] chapter 5, or other textbooks in this area.) The role of truth-values is played by certain “external” features of program behavior (e.g., the input-output relation); and the structure is the scheme by which programs, and program-components, are composed of smaller units.⁴ The area of programming languages offers also examples of non-denotational semantics, often referred to as *operational* (cf. [Winskel 1993] chapter 2). Actually, this is the older kind. It follows more closely the program’s dynamics, using modelings of computational processes (involving, for example, transitions between states) in order to specify the program’s behavior. There is a wide variety of modelings, some of which lead to branches of research rich in formal structure and in applications.⁵ The relevance of this fact, into which we need not go any further, is the very possibility of a non-compositional semantics.

Given that our aim is a systematic account of what makes sentences (types or tokens) true or false—or, more generally, what makes them relate in this or that particular way to the world—there seems to be no a priori reason why the account must take a compositional form. Compositional patterns, to be sure, cannot be ignored. It is hardly conceivable that we bypass the analysis of ‘The table is red’ into ‘The table’ and ‘is red’—viewed as a particular case of ‘*a is P*’. But this is still a far cry from an overall compositional semantics.

Non-compositional aspects derive from contextual dependencies, whereby the meaning of a phrase depends not only on its structure and the meanings of its components, but also on the embedding context. It is a fact about natural language that these aspects, which have been accorded ample attention by philosophers, have not been amenable to modeling at the same level of rigor attained in compositional semantics. In particular, when it comes to the effects of larger chunks of context, we find ourselves appealing to the special circumstances of the speaker’s activity, employing looser and vaguer explanations. In short, we take a pragmatic rather than a semantic tack. The present proposal shows that this need not always be so. More on the interplay of pragmatics and semantics is in the next section.

2. THE FORCE OF THE ANALYSIS

As a rule, the reactions to my proposed analysis of the two-line puzzle acknowledge its plausibility. But some have argued that other construals are plausible as well; in the end one finds oneself falling back on pre-theoretic intuitions. Let me therefore elaborate the analysis, with the aim of showing that it is quite compelling.

Liar sentences, or their equivalents, are used whenever statements that deny the truth of Liar sentences are made. As noted above, the same sentence (type) that produces a paradox is also used to make a true statement.⁶ We can trace the two uses, or two readings of the Liar sentence to two different ways of determining truth-values. The first consists in implementing (*) all the way, leading, in our case, to endless looping. The second way includes a higher level move: If the

evaluation of the sentence written in/on ... fails, because of loops (or because of other specified features), the failure is detected, following which ‘The sentence written in/on ... is true’ is classified as false and its negation as true. On the first reading, the Liar fails to state anything true or false. On the second, it states that it is not true on the first reading.

Some accounts leave out the second reading altogether. Kripke’s system [1975] represents only the first; consequently all Liar sentences and their kin lack truth-values. The second reading is seen by Kripke as a move beyond the system’s scope: to the metalanguage employed by the theoretician. The same is true, though less obviously so, of revisionary truth-theories (originally proposed by Gupta [1982] and by Herzberger [1982]). As in Kripke’s system, truth-values are assigned to sentence types. But the values are subject to revisions during the evaluation process and sometimes there is no convergence to a stable value. In particular, Liar sentences keep oscillating between ‘false’ and ‘true’. Yet the theoretician, who describes this very fact and who asserts that the Liar is not true (because it oscillates), makes an assertion that is true *tout court*—not an oscillating one. This metalinguistic perspective is beyond the system’s reach.⁷

Given that the second reading exists in actual usage, what determines how an utterance or an inscription is read? On my proposal, it is determined according to whether the token in question is or is not inside the loop. And this requires that ‘true’ and ‘false’ be construed as predicates over sentence-tokens. Otherwise we have to regard Liar sentences (types) as ambiguous sentences whose reading is determined by a host of circumstantial factors: the discussion within which the utterance or inscription takes place, what is known of the speaker’s intentions and knowledge, charity considerations, and so on. We fall back on informal pragmatics. Such a move is indeed common to all other accounts that have tried to grapple with the problem.⁸

There are two points in favor of my proposed analysis. The first is a direct argument for associating, respectively, the first and second readings of the Liar with the line 1 and line 2 tokens. The second is that the proposal substitutes formal rigorous criteria for informal pragmatic ones, criteria that generalize to a full-fledged system. Take the first point first. As just remarked, the non-truth of the sentence on line 1 is stated by using another token of the same sentence. Absent other considerations, there is no reason why it should not be the token on line 2. The assignment of ‘true’ to that token is therefore natural. Yet it is not forced. The token is not true if we decide, for whatever reason, to interpret it according to the first reading. But when it comes to the line 1 token there is no choice. Only the first reading applies. To read it in the second way is to read it as stating truly and successfully its own non-truth, and this is incoherent. If Jack says ‘What I am saying at this very moment is not true’ (and says nothing else), we can successfully and truly assert that he did not utter a truth: ‘What Jack said is not true’. But it is hardly conceivable that Jack’s utterance is true by virtue of its success in attributing non-truth to itself. Perhaps an utterance exemplifies failure, or *shows* its own failure, but it does not succeed in stating it. If it did it would not be a failure.

Further twists can be explored. Say, on a first reading the sentence on line 1 fails. Having detected the failure, one can read into the same token a statement of this fact. In that sense the sentence on line 1 is true after all. Some such view was put forward by Burge [1979]. But the implication of the story is that we should replace the sentence by two abstract entities, which represent the two rounds of thought, or the two acts of judgment. Indeed, pointers—as we shall see—are designed exactly to fulfill this role. In our case, however, the effect is smoothly achieved by attaching the different readings to the different tokens; which, moreover, accords with actual usage. Note also that the sentence on line 1 cannot be viewed merely as an ambiguous case—non-true under one reading, true under another. The reading under which it is true presupposes the reading under which it is not. Similarly, the sense, if there be such, in which Jack’s statement is true requires a previous sense in which it fails. But then “what Jack has said” encompasses failure; it is not a mere truth.

Pragmatics and Semantics. When it comes to interpreting actual speakers, pragmatic factors—specific considerations beyond semantic rules—are, of course, crucial. Contextual disambiguation is a necessary preparatory step to logical analysis; specific local factors can trump semantic and even grammatical norms;⁹ quantification ranges are, to a large extent, determined by implicit contextual parameters (it is rather the exception than the rule that ‘everyone’ is intended to range literally over all human beings). The story is well known and needs no repetition. It is nonetheless formal semantics which displays basic schemes of thinking that underlie natural language. The significance of ‘everyone’, ‘someone’ and their like is revealed by viewing them through the lens of quantificational logic.

The extent to which formal, or semi-formal, systems illuminate natural language changes with our theoretical knowledge. Linguistic patterns that are subject to loose surface descriptions may be later unpacked as vehicles of sophisticated logic. Compare, for example, the description obtained along the lines of Mill’s logic with the kind of analysis made possible by Frege. The latter system did not transform natural language into a formalism of sorts; but it brought into the open some of its underlying mechanisms. Since pragmatic factors will always play their crucial role, and since pragmatic accounts can be explanatory and revealing, there is a temptation to apply them across the board. The risk of this tendency is the substitution of easy hand waving for hard philosophical labor. The sea of language will not be harnessed, but any claimed piece of land enhances our understanding.

The following example, due essentially to Prior [1961], can illustrate the interplay of semantics and pragmatics in the case of Liar sentences. A student who thinks the teacher was mistaken in what he wrote on the blackboard next room—believed by her to be room 10—writes on the blackboard in front of her: ‘What is written on the blackboard in room 10 is not true’. As it happens, she got the room number wrong. She herself is in room 10. Here one might invoke Kripke’s [1976] distinction between semantic reference and speaker’s reference. On the semantic account, what she wrote fails, but—given sufficient evidence to warrant the story—we can attribute to her a different belief expressible by some paraphrase,

e.g., ‘What the teacher wrote on the blackboard in the next room is not true.’ (Note that no similar way out is available when a competent speaker says: ‘What I am now saying is not true.’) The semantics to be presented here does not address such pragmatic issues. Obviously, there is no systematic way of implementing charity considerations. And a speaker may fail by creating unwittingly a loop, without there being anything clear we can point to as “what the speaker had in mind”.

Some loops can be blocked by a systematic reinterpretation of the quantifiers. Let a book contain the sentence ‘Everything written in this book is true.’ If all other sentence-tokens in the book come out true, then this token creates a non-terminating loop; like the Truth Teller it sends us back to itself. Yet, we can plausibly interpret the writer as saying that all *other* sentences in the book are true. The loop is avoided by excluding the token from the range of the quantifier. This can be systematized into a rule: A token whose type contains a bounded variable that occurs under a semantic predicate (‘is true’, or ‘is false’) is excluded from the range of this variable. Such a rule can be incorporated into the formalism outlined in the sequel. I shall not be concerned here with this particular variant of the system.¹⁰ My point is only to show how a pragmatic consideration (“surely the writer did not intend to include this very same token among the sentences to which he refers”) can metamorphose into a semantic rule. The possibility is noteworthy because such a move is not available in general. ‘Everyone knows that in the summer the days are longer’—surely, the speaker does not mean literally that every human being on earth knows this, but only that every minimally educated, or minimally observant one does. *That* kind of consideration is not liable to semantic metamorphosis.

‘Pragmatics’ has been used here in a rather loose sense. Originally, the term covered everything not coming under syntax or under semantics, all the ways by which meaning is affected by context and special circumstance, which cannot be systematized with formal rigor. To a large extent the tradition is still with us. I have not attempted to sort this out and I have taken the informal contextual considerations to be those that bring in the speakers’ intentions. A more careful analysis should distinguish between the two. It is not an a priori truth that they must run together. (In non-linguistic domains, e.g., in decision theory, intentional subjective factors are fruitfully analysed through formal models.) But further discussion of this subject would take us beyond the scope of this work.

3. THE METHOD OUTLINED, POINTERS, PROPOSITIONS, AND RELATED MATTERS

Linguistic tokens are—to recap what was said above—physical objects that function in linguistic activities by virtue of the types—abstract syntactic constructs—which they call forth. Under “physical objects” are included also events, or physical processes, which occur at particular times in particular places. Physical parameters serve to distinguish between different tokens. But linguistic use enters into the identity conditions as well. In principle, the same physical object can be employed to represent different constructs in two different languages. What counts as a token is therefore the physical object *as it is used* in this or that language. Also mere physical similarity is neither sufficient nor necessary for the sameness

of the tokenized types. Having noted all this, let us take the token-type relation for granted.

The PTT evaluation procedure required that we include, as primitives of the system, certain operations on tokens that correspond to various syntactic operations on types. For example, an operation that correlates, with every token, p , of $\forall x\alpha(x)$ and with every substitutable constant term, t , a “token” (denoted as $p|t$) of the instantiation $\alpha(t)$. In general, such operations carry us out of the domain of tokens. Hence we had to generalize the concept of a token to that of a pointer: an object that points to a sentence (type). A token is a particular case of a pointer, it points to the type of which it is a token.

These additional primitive operations are no longer required. We assume only a non-empty set of objects (the pointers) and a function that associates with each of these objects a sentence-type. We can therefore make do with tokens. Nonetheless I shall argue that a generalized notion—that of a pointer—is called for. First let me give a rough outline of how truth-values are assigned, using tokens for that purpose.

We presuppose a fixed first-order language, \mathcal{L} . It is interpreted, except for two so-called *semantic predicates* $Tr(\)$ and $Fa(\)$, which take names of sentence-tokens as arguments. Assume, for simplicity, that every token has a name in \mathcal{L} , which is also its name in the metalanguage of this paper. Thus, if p is a token, $Tr(p)$ and $Fa(p)$ are sentences of \mathcal{L} . $Tr(p)$ says that p is true—or, if you want, expresses, or points to, a truth; $Fa(p)$ —that it is false.

Our goal is to interpret the predicates Tr and Fa ; that is, to determine their extensions. This is done by an *evaluation procedure*. Given any token, the procedure results either in assigning to it a truth-value, **T** (for ‘true’) or **F** (for ‘false’), or in classifying it as a failure. It is convenient to represent failure by an assignment of a third value, *GAP*. The evaluation procedure determines therefore a three-valued valuation of all tokens. The extension of Tr consists of the tokens that get **T**, that of Fa —of the tokens that get **F**. Tokens that get *GAP* are in neither extension. Hence,

$$\neg Tr(p) \wedge \neg Fa(p)$$

is true just when p gets *GAP*. This is the sentence that says in \mathcal{L} that p fails. The values **T** and **F** are referred to as *standard*.

The evaluation procedure is based on: (i) clauses that assign to tokens standard values, and (ii) clauses that determine failure, i.e., that assign *GAP*. The rules are applied recursively. At each stage we get a partial three-valued valuation.

Now, any partial three-valued valuation of the tokens induces a partial two-valued valuation of the sentences of \mathcal{L} . First, sentences not containing semantic predicates get their truth-values through the presupposed interpretation of the rest of \mathcal{L} . Next, if the token p gets a standard value, then $Tr(p)$ gets the same value and $Fa(p)$ gets the opposite value; if p gets *GAP* then both $Tr(p)$ and $Fa(p)$ get **F**. If p is unevaluated so are $Tr(p)$ and $Fa(p)$. Finally, this partial two-valued valuation is extended by using Kleene’s strong truth-tables (here the “third value” is not *GAP*, but ‘undefined’). Thus, if α is evaluated (is given a value by the induced partial valuation), then $\neg\alpha$ gets the opposite value, and if α is unevaluated so is $\neg\alpha$. The

conjunction $\alpha \wedge \beta$ gets **T** if both conjuncts get **T**; it gets **F** if at least one conjunct gets **F**; in all other cases it is unevaluated. And so on. Universal and existential sentences are handled by treating them, in the usual way, as infinite conjunctions and disjunctions. Call the partial two-valued valuation of sentences the *induced valuation*.

Alternatively, we can get the induced valuation by using the supervaluation method. This would constitute a different variant of our procedure.

The induced valuation of sentences is used, in its turn, to assign standard values to additional tokens, according to the following standard-values rule: *If the sentence α is evaluated and p is a token of it, then p gets the same value, provided that p has not been classified as a failure, i.e., provided that *GAP* has not been assigned to it at a previous stage.*

By applying recursively the standard-values rule, various tokens get standard values. Other tokens, which cannot be reached, remain unevaluated (e.g., both tokens on line 1 and on line 2 in the two-line puzzle). Such tokens are handled by rules that determine failure. One, which plays a central role, assigns *GAP* to unevaluated tokens that form so-called closed loops. Roughly speaking—the precise definition is given in section 4—a closed loop consists of unevaluated pointers that refer to each other through unevaluated sentences. Suppose, for example, that p is a token of $Fa(q) \vee \alpha$ and q is a token of $\neg Tr(p)$. This can be realized, e.g., as follows:

Line 3 Either the sentence on line 4 is false, or α .

Line 4 The sentence on line 3 is not true.

Here ‘the sentence on line 3’ is to be read as ‘the sentence-token on line 3’; it is a token-name that appears as ‘ p ’ in the formalization. Similarly, ‘the sentence on line 4’ appears as ‘ q ’. Let r be another token of $\neg Tr(p)$, say, on line 5.

The following diagram shows the parsing “trees” of the sentences in question. Nodes represent sentences. Nodes labeled by tokens (p , q , or r) represent the labeling token, as well as the tokenized sentence. (‘ α ’ is not such a label, but stands for the undisplayed part that arises from the sentence α .) As in a parsing tree, a node representing a sentential compound has, depending on the connective, one or two outgoing edges to nodes representing the main components; the connective labels the edges. (Quantified sentences may give rise to an infinite number of outgoing edges.) The difference between this and a usual parsing tree is the following. A node representing a sentence of the form $Tr(s)$ or $Fa(s)$ (where s is a token) has an outgoing edge (labeled by Tr , or Fa) to a node labeled by s . If, moreover, s labels already a node from which there is a path to the node in question, then the outgoing edge is to that node. Thus, our “trees” look like parsing trees, except that certain backward looping edges are allowed.¹¹

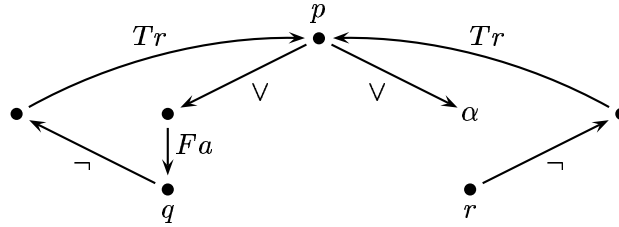


Figure 1

At some stage α gets a value under the induced valuation (if α does not contain semantic predicates, this stage is right at the beginning); then the tokens p , q , r , get their values. If α 's value is **F**, then p and q form a closed loop. Both get, via the closed loop rule, *GAP*; then r , which is not in this loop, gets **T**, via the standard-values rule. But if α has the value **T**, then $Fa(q) \vee \alpha$ gets **T**. Here there is no loop; p gets **T** and q and r get **F**. In general, a closed loop may contain any finite or infinite number of pointers. In the two-line puzzle, the line 1 token forms by itself a closed loop; it gets *GAP*, then the line 2 token gets **T**.

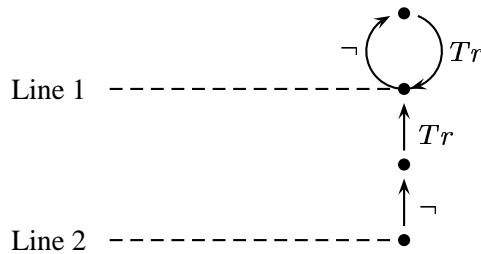


Figure 2

There is another rule for determining failure. It applies in cases where the valuation is not total and neither the standard-values rule nor the closed loop rule can be applied. Such cases involve, it can be shown, infinite descending chains. A simple example is an infinite sequence of tokens, each on a separate line, such that for every n , the sentence on line n is: ‘The sentence on line $n + 1$ is not true’. One can plausibly argue that, as far as tokens are concerned, we can ignore such cases. If we do this, we need only the standard-values rule and the closed loop rule. But infinite descending chains should be included within the general framework of pointers (this is discussed in section 5). Hence a third rule is added. When the recursive procedure terminates every token gets a value.

As the evaluation proceeds, additional sentences and additional tokens get standard truth-values and additional tokens are classified as failures. At the end we get a three-fold division of all tokens into true ones, false ones, and those that fail; and a two-fold division of all sentences into those that are true and those that are false. Tokens that do not fail get the same value as their tokenized sentences. It is, in principle, possible that all the tokens of some sentence (or, in the general setting, all pointers to it) fail. What the sentence says in the theoretician’s metalanguage cannot—in that case—be expressed by any token of a sentence of \mathcal{L} . Variants of the basic evaluation procedure are chosen with a view of minimizing this possibility.

Pointers. The setup does not require anything except the correlation between tokens and types. It is not even required that every sentence be tokenized. This generality indicates, however, that something more general is in play: a system that consists of a non-empty set and a function that correlates with each of its members a sentence. Further requirements, or additional structural elements, can be imposed later as the need arises. Some weighty arguments show the need for such a general concept.

For one thing, if we take tokens seriously, then their number—the number of inscriptions and utterances that have been and will be made by humans—is finite (unless the human race continues for infinite time). But we might want to have every sentence represented by some token, in which case we should speak of possible tokens rather than actual ones. There may be also technical reasons for going beyond tokens. As mentioned at the beginning of this section, the PTT system posited various operations that require something more general than tokens. But the more interesting reasons are not merely technical.

Think of the multifarious ways, besides utterances and inscriptions, which are used to communicate statements: all kinds of signaling, electronic mail, satellite communication, what have you. The general scheme that applies is this: Objects are used in acts of assertion, attempted assertion, or interpretation. The propositional content of what is asserted, or interpreted, is given through sentences that are associated with these objects. Yet the sentences, by themselves, may not be sufficient. Some other parameters of the object, e.g., its place in some global network, may affect what it expresses. What these objects are and how the associated sentences are determined varies according to case. The token-type relation is simple enough, but even there a closer look reveals a non-physical dimension at play. It would be pointless for the purpose of a general analysis to pin these objects down as physical entities. And it may not even be possible; what exactly is the “token” of a radio message that underwent scrambling and unscrambling?. As we shall see shortly, it is sometimes natural to include the sentences (types) as pointers to themselves. Later, in section 5, Gödel numbers serve as pointers to the sentences of arithmetic. Let us keep the possibilities open.

The meaning of the pointer system is given through its functioning: the way truth-values are assigned and failure is determined. In the context that concerns us here—that of a language containing its own truth-predicate—the pointer determines the metalevel at which the pointed to sentence is read. A pointer marks thereby a point of view.

Some Special Pointers. Sentences (types) can serve as pointers to themselves. In “‘Three is odd’ is true’ the sentence ‘Three is odd’ points to itself. At least, it is the best construal, because it means that ‘‘Three is odd’’ is interpreted as a name of the sentence, not of a pointer different from it.

This touches on the general question of how to construe the references of descriptive terms that appear, in English, as arguments of the predicates ‘true’ and ‘false’. Such terms are interpreted as referring to pointers, but little has been said so far about the identification of these pointers with familiar objects. Nor, for that

matter, is there a general prescription. The basic principle is that whenever a sentence expresses different things upon different uses (as in the two-line puzzle), the difference must be traced to a difference in pointers; therefore at least in one of these uses the pointer must differ from the sentence itself. If pointers are not readily available in the form of familiar objects they should be posited. This is not an ad-hoc move. The very purpose of pointers is to serve as pegs for different interpretations of the same sentence.

Subject to that constraint, it is desirable to construe the pointers so as to accord as much as possible with pre-theoretic usage. Since, in the two-line puzzle, what the sentence says differs from line to line, and since ‘the sentence-token on line ... is true’ is an acceptable phrasing, the obvious choice is to take the tokens as our pointers. The decisive factor here is not the term itself, but the predicate ‘true’ under which it occurs. Occurrences that are not under a semantic predicate can be construed as referring to sentence-types, e.g., ‘The sentence on line 1 has eight words.’

There is no self-referential sentence that refers to itself by enquoting, because the enquoted sentence is “part” of its name. Therefore, sentences that are pointers to themselves can be subject, in our formalism, to various constraints; e.g., the sentence α is different from any pointer p whose name occurs in α under a semantic predicate. More of this is in section 5.

Demonstrative pointers constitute another noteworthy kind. These are demonstrative acts that pick out sentences: ‘*This* [pointing to a sentence-token] is not true’. Upon formalization this sentence becomes $\neg Tr(p)$. On one construal, p is the pointed to token. But we can also construe p as that particular demonstration, assimilating thereby the act of pointing and the token into a demonstrative pointer that points to the sentence-type. Here, as well, constraints that preclude self-reference are plausible. Other examples of special kinds of pointers are in section 4, “Systems With Operations on Pointers”, and in section 5.

Propositions. It is tempting to say that by using a pointer one expresses, or tries to express, a proposition. I shall not resist the temptation. But my use of ‘proposition’ is innocuous, a suggestive way of putting things, not a commitment to autonomous entities. If we read $Tr(p)$ as ‘ p expresses a true proposition’, and $Fa(p)$ similarly, then $\neg Tr(p) \wedge \neg Fa(p)$ says that p does not express a true proposition and does not express a false proposition. Assuming that propositions are either true or false, this means that p does not express a proposition.

On the traditional view, a proposition is what a declarative sentence says and what is made true or false by the world. Propositions are front and center in the *Principia*, coming before language and enjoying an independent ontological status. Some philosophers, notably Quine and Davidson, have rejected propositions, either because they saw them as philosophical phantoms, or because they did not find them instrumental enough in the advancement of true understanding. Other positions, while not as sanguine as those of Russell or Ramsey, involve varying degrees of commitment to propositions. We need not go into the various views,

nor take a particular stand, since the present work does not presuppose this or that position.

Stalnaker [1972] and Lewis [1975] suggested that language be viewed as a device for pointing out propositions, where propositions are defined as sets of possible worlds. Language can indeed be perceived from a general perspective as a way of using objects to make statements. Yet the nature of these statements should itself be clarified through a language-oriented approach. What propositions are will come out of an account of truth-conditions, or of other external elements of linguistic usage. This circular clarification, whereby language is viewed as a tool for expressing propositions and propositions are what the analysis of language reveals, is the best we can do. While the framework of possible worlds is an invaluable instrument for clarifying modal notions, its imposition, in the form of preconceived propositions, forecloses the more significant uses of language-based analysis in the theory of meaning, the philosophy of logic, and in metaphysics.

So far for my view on propositions. Yet little in the present work hinges on it. Possible worlds do not come into the setup because we are not concerned here with modal notions. The system could be combined with a model of possible worlds, if the latter is needed to handle modalities.

I shall adhere to the tradition according to which the having of a truth-value ('true' or 'false') is a characteristic mark of propositions. This is more of a taxonomic convenience than a substantial position, since I admit aspects of meaning that do not require truth-values. The token on line 1, for example, is meaningful in as much as it sets off a sequence of moves in accordance with (*). Looping endlessly, the moves fail to yield a truth-value. But this very procedure constitutes the meaning—if you want, the Fregean sense—of that token. Let us therefore say that the token on line 1 has a sense, but fails to express a proposition. What this token suggests but fails to express is expressed by the token on line 2.

Since every sentence gets eventually a truth-value, every sentence expresses a proposition. But this is accomplished in the metalanguage, the one used for describing the evaluation procedure. If the sentence has a non-failing pointer to it, the proposition is expressed by that pointer; then, and only then, is it expressible in \mathcal{L} . We can thus say that every pointer points to a proposition: the one expressed (in the metalanguage) by the pointed to sentence. But a pointer *expresses* that proposition if and only if it gets a standard truth-value. Pointers that get *GAP* fail to express the propositions they point to.

The Strengthened Liar. In its original formulation the Liar sentence says of itself that it is false and the paradox consists in the fact that each of the assumptions, that the sentence is true and that it is false, leads to contradiction. A natural move is then suggested: the sentence is neither true nor false. The same move is available if the sentence says of itself that it is not true. Van Frassen [1968] proposed the so-called Strengthened Liar: if a sentence, which says of itself that it is not true, lacks a truth-value, then it is not true; but this is exactly what the sentence says, hence the sentence is true after all and the contradiction returns. The contradiction

returns also if the sentence says of itself that it is false: If it lacks a truth-value, then it is not false; but it says of itself that it is false, hence it is false after all.

The Strengthened Liar should be distinguished from the unable-to-say paradox, which consists of our being unable to say that the line 1 sentence is not true, without repeating this very same sentence. It is the latter—the subject of this work—that necessitates an attribution of truth-values to pointers.¹²

By declaring the Liar sentence (either type or token) a failure, we block the Strengthened Liar. For if the sentence fails to express a proposition it does *not*, contrary to appearance, say of itself that it is not true. We cannot argue that if it is not true then what it says of itself is true. There is nothing it says of itself—nothing, that is, whose truth-value is computable from its truth-conditions. The Strengthened Liar results from treating a token that fails as if it expressed something that can be evaluated for truth-value. If this move is disallowed, there is no paradox. Both the Liar and the Strengthened Liar are disposed of by casting Liar sentences as failed attempts to express propositions.

In many cases, failures can be salvaged; at least this follows from Russell’s view. ‘The king of France is bald’ fails because ‘The king of France’ fails to denote. But if we read the sentence as asserting, among other things, the existence of the king of France, then it succeeds in expressing a false proposition. The general strategy then is to assimilate, into the proposition, implicit presuppositions that are necessary for non-failure. Can something like this work for Liar sentences? A possible implicit presupposition is that the Liar sentence expresses a proposition. Indeed, Russell suggested that we include this as part of what the Liar sentence says. But the suggestion calls for quantifying over propositions. Short of this, we can have the presupposition that ‘The sentence on line 1 is not true’ is not written on line 1. But it is strange to read this as part of what the sentence says. Besides, what would be the implicit presupposition of a sentence that participates in a closed loop that involves many other sentences? Finally and conclusively, we would have to read Jack’s assertion of ‘What I am saying now is not true’ as ‘What I am saying now is not true and what I have just uttered is not ‘What I am saying now is not true.’ ’ Which is absurd.

4. THE METHOD IN FULL

A *pointer system* for a language \mathcal{L} consists of a non-empty set of objects called pointers and a mapping that associates with every pointer, p , a sentence of \mathcal{L} . We denote the associated sentence as:

$$p \downarrow$$

and say that p *points to* $p \downarrow$. No other structural elements or further requirements are needed. We need not even assume that every sentence is pointed to by some pointer (although this, and stronger requirements, will be later considered).

Assume a pointer system for \mathcal{L} , to be kept fixed throughout the discussion. As before, \mathcal{L} is a first-order language, whose vocabulary is interpreted, except for two semantic predicates, $Tr()$ and $Fa()$, which take as arguments pointer terms (expressions denoting pointers or variables ranging over them). Our goal is to extend the interpretation to the semantic predicates.

To simplify the presentation assume that the interpreted vocabulary consists of predicates and individual names. There are no function symbols. No loss of generality ensues, since the more general case is reducible to this in well-known ways. Operations on pointers, which form part of the pointer system (like those of PTT), may, however, call for special treatment and will be discussed later in this section.

Another simplifying assumption is that every member, a , of the universe in which \mathcal{L} is interpreted has a name in \mathcal{L} , which, moreover, we take to be ‘ a ’. Hence, we can speak of the sentence $\alpha(a)$ that instantiates the quantified sentence $Qx\alpha(x)$, whenever a is a possible value of ‘ x ’. This enables us to give a truth-definition for sentences, without going through assignments of objects to free variables. There is no loss of generality, since the handling of the more general case is derivable as a mere technicality. We use

$$‘p’, ‘q’, ‘r’, ‘p_1’, ‘q_1’, ‘r_1’, ‘p’’, \dots$$

as metavariables ranging over pointers. We may therefore speak of the pointer p and of the atomic sentence, $Tr(p)$, of \mathcal{L} .

At each stage of the evaluation process we have a partial function defined over pointers, assigning to each pointer in its domain either **T** or **F** (the standard values), or *GAP*—which, as before, signifies failure. Henceforth, the unqualified term ‘valuation’ means a partial three-valued valuation over pointers. Valuations that are everywhere defined are *total*. We shall use

$$‘v’, ‘u’, ‘v_1’, ‘u_1’, ‘v’’, ‘u’’, \dots$$

for valuations. A pointer, p , is *evaluated* by v if $v(p)$ is defined, and is *unevaluated* otherwise. The equality $v(p) = w$ means that p is evaluated by v and its value is w . Thus, $v(p) \neq w$ means that either p is unevaluated by v or p is evaluated and its value is different from w .

As noted already our valuations induce two-valued (partial) valuations over sentences. The valuation induced by v is denoted as \bar{v} and referred to as the *the induced valuation*. Here, in more detail, are its defining clauses:

- (1) If α is an atomic sentence not containing a semantic predicate, then $\bar{v}(\alpha)$ is α ’s truth-value in the given interpretation of \mathcal{L} .
- (2) If $v(p) = \mathbf{T}$ then $\bar{v}(Tr(p)) = \mathbf{T}$ and $\bar{v}(Fa(p)) = \mathbf{F}$.
- (3) If $v(p) = \mathbf{F}$ then $\bar{v}(Tr(p)) = \mathbf{F}$ and $\bar{v}(Fa(p)) = \mathbf{T}$.
- (4) If $v(p) = \text{GAP}$ then $\bar{v}(Tr(p)) = \bar{v}(Fa(p)) = \mathbf{F}$.
- (5) For non-atomic sentences, \bar{v} is determined according to Kleene’s strong truth-tables, where \forall and \exists are construed as (infinite) conjunctions and disjunctions.

The last clause, recall, amounts to applying recursively the following clauses:

- (5.1) If $\bar{v}(\alpha)$ is defined, then $\bar{v}(\neg\alpha) = \neg\bar{v}(\alpha)$, where $\neg\mathbf{T} = \mathbf{F}$ and $\neg\mathbf{F} = \mathbf{T}$.
- (5.2) If $\bar{v}(\alpha) = \bar{v}(\beta) = \mathbf{T}$, then $\bar{v}(\alpha \wedge \beta) = \mathbf{T}$.
- (5.3) If either $\bar{v}(\alpha) = \mathbf{F}$ or $\bar{v}(\beta) = \mathbf{F}$, then $\bar{v}(\alpha \wedge \beta) = \mathbf{F}$.
- (5.4) If $\bar{v}(\alpha(a)) = \mathbf{T}$ for all a in the range of ‘ x ’, then $\bar{v}(\forall x\alpha(x)) = \mathbf{T}$.

- (5.5) If $\bar{v}(\alpha(a)) = \mathbf{F}$ for some a in the range of ‘ x ’, then $\bar{v}(\forall x\alpha(x)) = \mathbf{F}$.
- (5.6) The clauses for other connectives and for existential quantification are obtained by expressing them in terms of \neg , \wedge and \forall .

A different way of inducing a valuation over sentences—which leads to a variant of the procedure—uses, instead of (5), the supervaluation method: A sentence α is given a value, if it has this value under all total valuations of the atomic semantic sentences that extend the valuation determined by (1)-(4). The use of supervaluations at this point does not necessitate changes in the other parts of our evaluation procedure. The system is fully modular. The phrasing of the rules and of the main results remains the same.

\emptyset is the empty valuation (i.e., all pointers are unevaluated under \emptyset). Note that $\bar{\emptyset}$ is defined for all sentences not containing a semantic predicate; it assigns to each its truth-value under the given interpretation of the non-semantic predicates and the quantifier ranges.

$u \subseteq v$ means that v is an extension of u . Obviously, $u \subseteq v$ implies $\bar{u} \subseteq \bar{v}$. If v is a total valuation, then \bar{v} is a total valuation of all sentences, which determines an interpretation of the whole of \mathcal{L} , extending the given interpretation of its non-semantic part.

The procedure is based on rules, whose application to a given valuation, v , yields a valuation v' . In general v' need not be an extension of v . Values can be revised. But if v is a “good” valuation (to be defined in the sequel) then v' extends v and is moreover itself “good”. The class of such valuations is also closed under unions of ascending chains. If we start with a “good” valuation—in particular, if we start with \emptyset —we will eventually get a total valuation over pointers that has the desired properties.

There are three rules altogether. The first is (SV)—a rule that assigns standard values. The other two are failure rules, which assign *GAP*.

- (SV) If $p \downarrow = \alpha$, $\bar{v}(\alpha)$ is defined and $v(p) \neq \text{GAP}$, then assign to p the value $\bar{v}(\alpha)$.

The sense of (SV) is obvious: any pointer p to a sentence α should get the same truth-value as α , unless this pointer has been declared already a failure. (Recall that $v(p) \neq \text{GAP}$ means that either $v(p)$ is standard or it is undefined.)

The condition that $\bar{v}(p \downarrow)$ is defined and $v(p) \neq \text{GAP}$ is the *enabling condition* for the rule (SV). It is necessary and sufficient for the applicability of (SV) to the pointer p . If it obtains we say that the rule is *enabled by v on the pointer p* . Each of the other rules has its enabling condition and the same terminology is used.

If p is unevaluated, the application of (SV) to p results in an extension of v . If $v(p)$ is evaluated and standard, the application may, in principle, revise this value. But if $v(p) = \text{GAP}$, (SV) is not enabled; hence *GAP* cannot be revised. We shall later consider another rule for assigning standard values, which makes it possible, in certain cases, to revise *GAP*. If, however, we start with a “good” valuation, there

will be no revisions. As far as “good” valuations are considered, we need only (SV) for assigning standard values.

Rules that Determine Failure. Roughly speaking, *GAP* is assigned in situations where standard values cannot be assigned by repeated applications of (SV). We try to pin the failure on a small number of pointers, so as to leave other pointers operative. Some examples and a preliminary discussion serve to motivate the definitions that follow.

Example 1 Let $p \downarrow = Tr(q) \vee Fa(r)$, $q \downarrow = \neg Tr(p)$, $r \downarrow = \neg Tr(p)$.

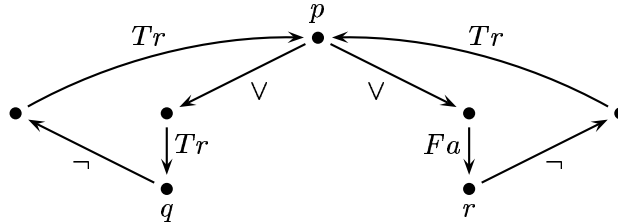


Figure 3

If p, q, r are unevaluated by v , then we cannot assign to any of them a value by repeated applications of (SV); p send us to q and to r and each of q and r sends us back to p . They form a closed loop, into which there is no breaking. The closed loop rule will assign to each the value *GAP*.

Example 2 Let $p \downarrow = Tr(q) \vee (Fa(r) \wedge (0 = 1))$, $q \downarrow = \neg Tr(p)$, $r \downarrow = \neg Tr(p)$.

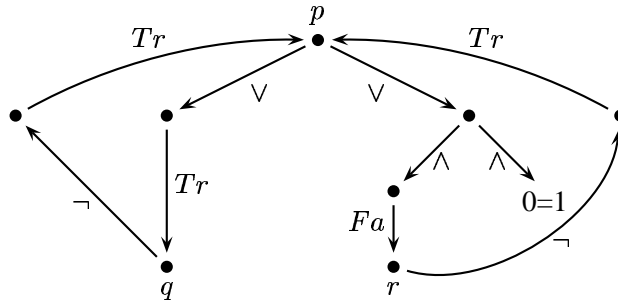


Figure 4

Here, again, p, q, r seem to send us to each other. But $\bar{v}(Fa(r) \wedge 0 = 1) = \mathbf{F}$, independently of r 's value. This leaves only p and q in the closed loop. Each gets *GAP*. Then r gets **T**.

Assume that p is unevaluated by v and that $p \downarrow = \alpha$. (SV) is enabled on p iff α is evaluated (by \bar{v}). Assume that α is unevaluated. If α is non-atomic, it is either a sentential compound or a quantified sentence ($\forall x\beta$ or $\exists x\beta$). In the first case α must have an unevaluated immediate sentential component;¹³ in the second case— an unevaluated instantiation, $\alpha(a)$. If α is atomic, it is either $Tr(q)$ or $Fa(q)$, where q is unevaluated by v (other atomic sentences are evaluated by $\bar{\emptyset}$, hence by \bar{v}). This shows that if (SV) is not enabled on an unevaluated pointer, p , then, in order to enable it, other unevaluated pointers should be given values. We can imagine that

the evaluation of p proceeds by calling these other pointers. Each of these may call pointers, in its turn, and so on.

Definition (I) A pointer p *calls directly* a pointer q , under a given valuation v , if p is unevaluated and there is a sequence of sentences $\alpha_0, \dots, \alpha_n$, unevaluated by \bar{v} , such that (i) $p \downarrow = \alpha_0$, (ii) for every $i < n$: α_{i+1} is either an immediate sentential component of α_i , or an instantiation of it and (iii) α_n is either $Tr(q)$ or $Fa(q)$.

(II) p *calls* q (under v), if there is a sequence of pointers p_0, \dots, p_n , such that $p_0 = p$, for every $i < n$, p_i call directly p_{i+1} (under v), and $p_n = q$.

Note that calling under v implies calling under any u such that $u \subseteq v$. In particular it implies calling under \emptyset .

In Example 1, each pointer calls, under \emptyset , itself and the other pointers. In Example 2, p does not call r , because the sentence $Fa(r) \wedge (0 = 1)$ is evaluated. Indeed, r 's value is not required for assigning a value to p .

In the sequel we shall omit references to the valuation v , when this valuation is understood from the context. Thus, if v is given, we shall say that (SV) is enabled on p , meaning that it is enabled by v ; or that p calls q , meaning that it calls it under v .

Lemma 1 If p is unevaluated by v , then (SV) is not enabled on p iff p calls some pointer.

The “if” part is trivial, since if p calls some pointer then $p \downarrow$ is unevaluated by \bar{v} . The “only if” part follows from the observations above and the fact that p calls some pointer iff it calls directly some pointer.

Definition A set of pointers X is *closed* for a given valuation, v , if it consists of unevaluated pointers and every unevaluated pointer called by a member of X is in X . If, in addition, every member of X calls some member of X , then X is said to be a *closed non-terminating* set for v .

(If p calls, under v , q , then q is unevaluated because the last sentence in the sequence that reaches to q is $Tr(q)$ or $Fa(q)$ and this sentence is unevaluated by \bar{v} . The last definition was phrased so as to apply also to another variant, mentioned later (“Systems With Operations on Pointers”), in which certain calls can go to evaluated pointers.)

Lemma 2 Assume that X is closed for v . Then (SV) is not enabled on every member of X iff X is non-terminating. In particular, (SV) is not enabled on every unevaluated pointer iff the unevaluated pointers form a closed non-terminating set.

Proof The first claim follows immediately from Lemma 1. The second—from the first, by observing that the set of all unevaluated pointers is closed.

Definition A set of pointers, X , is a *closed loop*, for a given valuation v , if it is closed and every pointer of X calls itself and every other pointer of X .

Obviously, a closed loop is a closed non-terminating set. Our first failure rule assigns *GAP* to all members of such a set:

Closed Loop Rule If X is a closed loop for v , assign *GAP* to every pointer in X .

Note: The closed loop rule is enabled by v on each of the members of the closed loop. Its application to any pointer in the loop results in assigning *GAP*, in a single step, to *all* the closed loop members. Since these are unevaluated pointers, the application extends v .

Sometimes the failure of pointers is not traceable to closed loops. Call a non-empty set of pointers *groundless under v* , if it is a closed non-terminating set which does not have any non-empty subset that is a closed loop. The simplest example of a groundless set (under the empty valuation) is an infinite descending chain:

$$(DC1) \quad p_0 \downarrow = Tr(p_1), \quad p_1 \downarrow = Tr(p_2), \dots, \quad p_i \downarrow = Tr(p_{i+1}), \dots$$

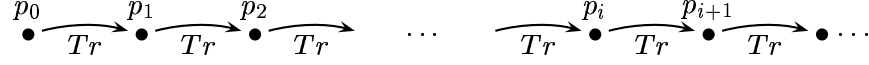


Figure 5

Groundless sets can be very intricate, but they all involve, in this way or another, infinite descending chains. This follows from the lemma below, whose proof, though not very difficult, is omitted.

Lemma 3 If X is a groundless set (under v) then there is an infinite sequence of members of X : $p_0, p_1, \dots, p_n, \dots$, such that each p_i calls p_{i+1} and $p_m \neq p_n$ whenever $m \neq n$.

Consequently, every groundless set is infinite. Closed loops, on the other hand, can be either finite or infinite. Note that all closed loops contained as subsets in a closed set must constitute terminal clusters of the downgoing chains. In a groundless set there are no such terminal clusters.

Call a pointer p *groundless* (under v) if it is a member of some groundless set. This, as is easily seen, is equivalent to the condition that p and all unevaluated pointers called by p , constitute a groundless set. Groundless pointers cannot be given values by repeated applications (to unevaluated pointers) of either (SV) or the closed loop rule. We handle them through our last rule:

Groundless Pointers Rule If there are pointers that are groundless under v , assign to all of them *GAP*.

As in the case of the closed loop rule, this rule is enabled on any groundless pointer and its application assigns *GAP* to all of them.¹⁴

Starting with the empty valuation \emptyset , we apply the rules as long as this is possible, taking unions at limit ordinals. In this way we get an increasing sequence of valuations:

$$v_0, v_1, \dots, v_\lambda, \dots$$

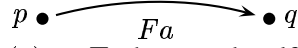
whose union is the desired total valuation. It is easily seen that any given valuation enables on any pointer no more than one rule. But we usually have considerable freedom in choosing where to apply a rule. The main result states that the final valuation does not depend on the choices made in passing from v_λ to $v_{\lambda+1}$. This is also true if v_0 is any “good” valuation, i.e., the final outcome depends only on v_0 .

Self-Supporting Valuations and the Main Result. The above mentioned “good” valuations are now to be defined. They are characterized by the general term *self-supporting*.

Definition A valuation v is *weakly self-supporting* if, for every evaluated p , if $v(p)$ is standard, then $\bar{v}(p \downarrow)$ is defined and $v(p) = \bar{v}(p \downarrow)$.

Weak self-support means that v supports, via \bar{v} , its own assignments of standard values. Consider, for example, two pointers p and q such that:

$$p \downarrow = Fa(q)$$



If we have: $v(p) = \mathbf{T}$, $v(q) = \mathbf{T}$, then weak self-support is violated, because $\bar{v}(p \downarrow) = \mathbf{F}$. The same is true if $v(p) = \mathbf{T}$, $v(q) = \text{GAP}$. In both cases the value of p can be revised to \mathbf{F} by applying (SV). Weak self-support is also violated if $v(p)$ is standard and q is unevaluated. But weak self-support is not violated if $v(p) = \text{GAP}$, no matter what q 's value is.

Obviously, the total valuation that constitutes the semantics should be weakly self-supporting: If p has a standard value, this should coincide with the value of the pointed to sentence.

We now consider a rule, (SV'), by which certain assignments of GAP to p are revisable to the induced value of $p \downarrow$. First, note that an assignment of GAP obtained by applying a failure rule should *not* be revisable. Say p gets GAP via a failure rule. Then p and all unevaluated pointers called by p , through unevaluated sentences, get GAP . As a result of these new assignments the previously unevaluated $p \downarrow$ gets a standard value. But we do not want to revise the assignment of GAP to p . Hence, the assignment of GAP to p should not be revisable to $p \downarrow$'s value, if the latter derives, roughly speaking, from assignments of GAP to pointers called by p .

Let the *standard part* of v , denoted v^s , be the restriction of v to those pointers whose values under v are standard: $v^s(p) = w$ iff $v(p) = w$ and w is \mathbf{T} or \mathbf{F} .

(SV') If $p \downarrow = \alpha$ and $\bar{v}^s(\alpha) = w$, then assign to p the value w .

Using (SV'), an assignment of GAP to p can be revised to $\bar{v}(p \downarrow)$, provided that this latter value derives from standard values assigned by v . If p gets GAP via a failure rule, then $p \downarrow$'s value does not derive from the standard part of v (if it did, $p \downarrow$ would have been evaluated already and the failure rule would not have applied). Hence (SV') cannot revise assignments made by failure rules.

Definition v is *strongly self-supporting* if it is weakly self-supporting and, for every p , if $v(p) = \text{GAP}$, then there is no extension of v under which this assignment can be revised, via (SV'). Spelled out fully, the second condition is: If $v(p) = \text{GAP}$, then for every u such that $v \subseteq u$, $\bar{u}^s(p \downarrow)$ is undefined.

Assume, for example:

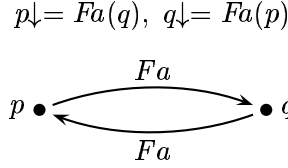
$$p \downarrow = \neg Tr(q)$$

If p gets GAP , then q must get GAP in order to satisfy strong self-support. (If q gets a standard value, p 's GAP is revisable via (SV'), and if q is unevaluated, then p 's GAP is revisable when we extend the valuation by assigning to q a standard value.)

We might, or might not want to impose strong self-support. If, in the last example, we find valuations that assign GAP to p and a standard value to q unacceptable, we can rule them out by adopting strong self-support. But we might want to allow the possibility that p fails—for some reason—even though q gets a standard value. We might also want to leave open the possibility of assigning—for some

reason— GAP to p before evaluating q . In this case weak self-support is all we require.

If $p \downarrow = \neg Tr(p)$ then any weakly self-supporting valuation defined for p must assign to it GAP . The same is true if $p \downarrow = Fa(p)$. But if $p = Tr(p)$, there is a strongly self-supporting valuation that assigns to p GAP , another that assigns to it \mathbf{T} , and another that assigns to it \mathbf{F} . This marks the difference between the Liar and the Truth Teller. To take another example, let



There is a strongly self-supporting valuation that assigns the value GAP to both, one that assigns \mathbf{T} to p and \mathbf{F} to q , and another that assigns \mathbf{F} to p and \mathbf{T} to q . There are two more weakly self-supporting valuations that evaluate both pointers. One assigns GAP to p and \mathbf{F} to q , the other— \mathbf{F} to p and GAP to q .

In what follows, ‘self-supporting’ can be read throughout either as ‘weakly self-supporting’ or as ‘strongly self-supporting’. All the results hold under each of these readings.

Lemma 4 (I) If v is self-supporting and if v' is obtained from v by applying either (SV) or a failure rule, then v' is self-supporting.

(II) If $\{v_i\}_{i \in I}$ is a family of self-supporting valuations directed under inclusion (that is, for every $i, j \in I$ there is $k \in I$, such that $v_i \subseteq v_k$ and $v_j \subseteq v_k$), then their union, $\cup_{i \in I} v_i$, is self-supporting.

Given any self-supporting valuation we can get, by applying repeatedly our three rules, an increasing chain of self-supporting valuations, where each valuation extends the previous ones. At limit ordinal we take unions and go on. As long as there are unevaluated pointers we can continue. For if (SV) cannot be applied to any unevaluated pointer, then, by Lemma 2, the unevaluated pointers form a non-terminating closed set. If there are closed loops, the closed loop rule applies, else the groundless pointers rule does.

Theorem 1 Every self-supporting valuation, v , can be extended to a total self-supporting valuation, v^* , by constructing a sequence:

$$v_0, v_1, \dots, v_\lambda, v_{\lambda+1}, \dots, v_\nu$$

such that: $v_0 = v$, $v_\nu = v^*$, for each λ , $v_{\lambda+1}$ is obtained from v_λ by applying an enabled rule, and, if λ is a limit ordinal > 0 , then $v_\lambda = \cup_{\mu < \lambda} v_\mu$. All such sequences that start with v end with the same total valuation. Moreover, the closed loops that are produced in the course of the process are the same in all sequences.¹⁵

Note: In other setups, such a result is proved by the minimal-fixed-point technique: An operator O , which transforms valuations to valuations, is said to be monotone if $u \subseteq v$ implies $O(u) \subseteq O(v)$. It can be shown that if O is monotone and $v \subseteq O(v)$, then there is a valuation v^* that extends v , which is a *fixed point* (i.e., $O(v^*) = v^*$), such that every fixed point that extends v extends v^* . This so called *minimal fixed point* is obviously unique. The same applies to a family of monotone operators. If

each of them extends the initial v , then there is a minimal extension of v that is a fixed point for every operator in the family. In our case define, for each pointer p , an operator O_p as follows:

If v enables some rule on p , then $O_p(v)$ is the valuation obtained from v by applying that rule to p . Else, $O_p(v) = v$.

Since at most one rule is enabled on p , the definition is legitimate. Had the O_p 's been monotone, we could have used the fixed point argument. But they are not. A valuation u might enable a rule on p , while an extension of it might not enable any rule, or might enable a rule that assigns to p a different value. The proof of Theorem 1 is therefore more intricate. It uses, in a way, a minimal-fixed-point approach; but it is based on a partial ordering different from \subseteq , which is defined, moreover, by means of the operators O_p . The main steps consist in showing that any ascending chain in this partial ordering has a least upper bound, which is the usual union, and that the operators are monotone with respect to it.

Note: Since the closed loops are determined solely by the initial v , we can classify the pointers that get *GAP* under v^* , into *closed-loop pointers*—which get *GAP* via the closed loop rule, and *groundless pointers*—which get it via the groundless pointers rule.

In the basic variant of our proposal, the semantics is given by the total self-supporting valuation determined by \emptyset . Other total self-supporting valuations can be used to distinguish the Truth Teller from the Liar, or to make various other distinctions into which we shall not go here. Henceforth, unless otherwise stated, the semantics is that of the basic variant, i.e., the initial valuation is \emptyset .

Recall that we also have the choice of using the supervaluation method, instead of Kleene's strong truth-tables, in determining the induced valuation. For example, if $p \downarrow = Tr(p) \vee \neg Tr(p)$ ('What I am saying now is either true or not true'), then in the Kleene-based assignment p gets *GAP*, but in the one based on supervaluations it gets **T**. All the main results hold for the supervaluation variant.

Expressibility Requirements. An important desideratum of the setup is the expressibility of many propositions by pointers. The general goal is that, given any sentence of \mathcal{L} , there be a non-failing pointer to it. Call this the *general expressibility requirement*. A more specific desideratum, is that, for a given p , there be non-failing pointers to $Tr(p)$, $Fa(p)$, $\neg Tr(p)$, $\neg Fa(p)$, and to $\neg Tr(p) \wedge \neg Fa(p)$. Using these pointers we can express directly all the information concerning p 's truth-value. Call this the *local metalevel requirement*.¹⁶

Some natural assumptions about the availability of pointers must be made. So far we did not even assume that there are pointers to every sentence. If this is assumed, then the local metalevel requirement for p holds whenever p has a standard value. This is implied by the following theorem. Here and henceforth in the discussion, the presupposed valuation is the total valuation that determines the semantics, unless the text indicates otherwise.

Theorem 2 If every pointer called directly by q , under the valuation \emptyset , has (in the final valuation) a standard value, then q has a standard value.

Proof Assume, for contradiction, that q gets *GAP*. It must get it at some stage via a failure rule. Let v be the valuation at that stage. Then q is a member of some closed non-terminating set (for v), say X , all of whose members get at that stage *GAP*. Since X is non-terminating, it contains a pointer called directly by q under v —hence also under \emptyset . Therefore some pointer called directly by q under \emptyset gets *GAP*. Contradiction.

Corollary If p_1, \dots, p_n have standard values, then any pointer that points to a sentential compound of $Tr(p_i)$'s and $Fa(p_i)$'s, $i = 1, \dots, n$, has a standard value.

In particular, if p does not fail, so do the pointers, which by assumption exist, to $Tr(p)$ and to $Fa(p)$ and to all their sentential compounds. But the local metalevel requirement is not guaranteed if p fails. If p is the line 1 token, then p is a pointer to $\neg Tr(p)$. Since it fails, we need an additional pointer. Moreover, there is, for every n , a simple example showing that n pointers will not suffice. Let Jack say: ‘What Ann says is true, and what Beth says is true, and ...and what Zina says is true’, and let each of the women say: ‘What Jack says is not true’. Then all the utterances fail, via the closed loop rule. We need an additional pointer to ‘What Jack says is not true’.

Let us therefore assume that each sentence has an infinite number of pointers to it. The infinity here is “potential” rather than “actual”; given any sentence, α , and any finite set of pointers, there should be a pointer to α outside this set. With this assumption, there is a rich class of propositions expressible by pointers.

Let $r \downarrow = \alpha$ and consider the set of all pointers called, under \emptyset , by r . It is easily seen that the set depends on α only. Say that p is *involved in the sentence* α if it belongs to this set.¹⁷ Obviously, the set of pointers involved in α is closed for \emptyset . Note that only the pointers in this set have to be considered in order to determine α 's value. (This set is empty iff α is evaluated by $\overline{\emptyset}$.)

Example: Let $\alpha = \forall x[\beta(x) \rightarrow \gamma(x)]$, where $\beta(x)$ does not contain semantic predicates. A pointer is involved in α iff it is involved in some $\gamma(a)$, such that $\overline{\emptyset}(\beta(a)) = \mathbf{T}$. (For if $\beta(a)$ gets \mathbf{F} , the instantiation $\alpha(a)$ gets \mathbf{T} , vacuously). If the set of all a 's for which $\overline{\emptyset}(\beta(a)) = \mathbf{T}$ is finite and each of the corresponding $\gamma(a)$'s involves a finite number of pointers, then the total number of pointers involved in α is finite.

Theorem 3 If α involves a finite number of pointers, then there is a non-failing pointer to α .

Proof Let X be the set of pointers involved in α and let q be any pointer to α not in X (there is one by our assumption). Start the evaluation process by applying rules to members of X only. Consider a stage at which the set, Y , of unevaluated pointers of X is not empty and (SV) is not enabled on any of them. Let v be the valuation at that stage. An unevaluated pointer that is called by a member of Y under v is also called under \emptyset ; since X is closed for \emptyset the pointer is in X , hence it is in Y . Therefore Y is closed for v . Since (SV) is not enabled, Y is non-terminating. Since X is finite, Lemma 3 implies that Y contains a closed loop as a subset. We can apply the closed loop rule, assigning thereby *GAP* to some members of X , and go on. This shows that all members of X can be evaluated without evaluating any

pointer outside X . At that stage α is evaluated and we can assign to q its value by applying (SV).

Theorem 3 implies that the local metalevel requirement is satisfied for all p 's that point to sentences involving a finite number of pointers. Because if $p \downarrow$ involves a finite number of pointers, so does any sentential compound, α , of $Tr(p)$ and $Fa(p)$ (the pointers involved in α consist of p and the pointers involved in $p \downarrow$).

The assumptions of Theorem 3 are satisfied in any situation in which a finite number of people make statements about the world and about statements made by people within this group. Hence, if p is any utterance made by a group member, there is a pointer q to the same sentence—say, an utterance of $p \downarrow$ made by an outsider—that does not fail.

Note that the finiteness of X (the set of the pointers involved in α) is needed at two points: (i) for the existence of a pointer to α , which is not in X , (ii) to ensure that X does not contain groundless pointers. (If X contains groundless pointers, an application of the groundless pointers rule may assign *GAP* to pointers outside X , possibly to all pointer that point to α .) The finiteness of X is not needed if (i) and (ii) are presupposed. Consequently we have: *If p is not involved in $p \downarrow$ and none of the pointers involved in $p \downarrow$ is groundless, then p does not fail.* This indicates that the general expressibility requirement may not obtain because of two reasons: (i) the existence of groundless pointers, (ii) the existence of a closed loop that contains all the pointers to some sentence. Take the first reason first.

Groundless Pointers and Black Holes. It can be shown that if $p \downarrow = q \downarrow$ and one of the two pointers is groundless, so is the other.¹⁸ Hence, if p is groundless, the proposition pointed to by p is not expressible by any pointer.

Also, if p is groundless so are all pointers to $Tr(p)$, $Fa(p)$, to their negations, and to $\neg Tr(p) \wedge \neg Fa(p)$. Hence we cannot assert in \mathcal{L} that p is not true, nor that it is not false, nor that it is a gap. In fact, p is what in PTT was called a *black hole*. The definition of holes and black holes for the present system is as follows.

A pointer p is a *hole* if it is a gap, and every pointer to a sentential compound, α , of $Tr(p)$ and $Fa(p)$ that conveys non-trivial information about p 's value is a gap. (Conveying non-trivial information means that the two-fold division induced by α 's two possible truth-values determines a non-trivial division in the set of p 's three possible values.) This generalizes. Call a sentence, α , *$n+1$ -informative about p* if there is a sequence of pointers p_0, p_1, \dots, p_n , such that: (i) $p_0 = p$ and for each $k > 0$ $p_k \downarrow$ is a sentential compound of $Tr(p_i)$'s and $Fa(p_i)$'s, $i = 0, \dots, k-1$, (ii) α is a sentential compound of $Tr(p_i)$'s and $Fa(p_i)$'s, $i = 0, \dots, n$, and (iii) α conveys non-trivial information about p 's value. Then p is an *n -hole*, where $n > 0$, if every pointer to an n -informative sentence about p is a gap. A black hole is an n -hole for all n . Every groundless pointer is a black hole. There can, however, be black holes that are not groundless.

Large Closed Loops. This is a more interesting and, I shall argue, a more serious concern than groundless pointers. The latter derive essentially from infinite descending chains—a feature that the pointer system may or may not have. Large closed loops, on the other hand, depend on the expressive power of \mathcal{L} . Assume

that \mathcal{L} comprises an apparatus for defining recursive sets and functions, sufficient for describing its own syntax. Assume moreover that the pointing relation between pointers and sentences is describable in \mathcal{L} . This implies that well-formed formula such as $x \downarrow = \alpha$ or $x \downarrow = p \downarrow \vee y \downarrow$ (where ‘ x ’ and ‘ y ’ range over pointers, p is a pointer and α is a sentence) can be translated into \mathcal{L} . Now suppose that p is a pointer such that:

$$(L1) \quad p \downarrow = \forall x [x \downarrow = p \downarrow \rightarrow \neg Tr(x)]$$

If pointers are tokens, the sentence in (L1) can be realized as:

(L1') Anyone who says what I am saying now does not utter thereby a truth.

Then p fails and so does every pointer to $p \downarrow$. Or consider:

$$(L2) \quad q \downarrow = \forall x [x \downarrow = \neg Tr(q) \rightarrow \neg Tr(x)]$$

(L2') Anyone who says that what I am saying now is not true does not utter thereby a truth.

Then q fails and so does every pointer to $\neg Tr(q)$. Still q is not a full-fledged hole (if $r \downarrow = Fa(q)$ then r gets **F**) and it is a long way from a black hole (if $r \downarrow = Tr(q)$ and $r' \downarrow = \neg Tr(r)$, then r' get **T**; in this way we can say indirectly that q is not true). But holes and even black holes can be constructed along these lines. The constructions are quite involved and there are no English paraphrases, unless “English” is extended by a technically defined vocabulary. In section 5 a variant of the evaluation procedure is presented, which ensures—at some price—the local metalevel requirement. There are other variants, in which pointers such as p and q above are “penalized” by early assignments of *GAP*. They will not be discussed in this paper.

Systems With Operations on Pointers. The PTT system includes among its primitives various operations on pointers. We mentioned already the function $()|()$, which associates, with every pointer p to a quantified sentence $Qx\alpha(x)$ and every substitutable constant term t , a pointer $p|t$ to $\alpha(t)$. Another pair of operations, written in suffix notation as $()1$ and $()2$, associate, with every p that points to $\alpha \wedge \beta$ or to $\alpha \vee \beta$, a pointer $p1$ to α , and a pointer $p2$ to β . Furthermore, if p points to $\neg\alpha$, then $p1$ points to α and $p2 = p1$. (In other cases, $p1$ and $p2$ are defined by arbitrary stipulation: $p1 = p2 = p$. Similarly for $p|t$.)

Operations on pointers can be added to the present system, without changing the evaluation procedure. But the operations are significant only if they enter into the procedure so as to make a difference. In the PTT system, if p points to a conjunction, it gets, in the final evaluation, **T** if both $p1$ and $p2$ get **T**; it gets **F** if at least one of them gets **F**; and it gets *GAP* in all other cases; This is not true if we add the operations to the PTP system (the system of the present work), without changing its evaluation rules. Consider, for example:

Line 6 Snow is white and the second conjunct of the sentence on line 6 is not true.

Let p be the token on line 6. Then we have:

$p1\downarrow = \text{'Snow is white.'}$
 $p2\downarrow = \text{'The second conjunct of the sentence on line 6 is not true.'}$

We can plausibly identify $p1$ and $p2$ with *subtokens* of p : the first is the token on line 6 of ‘Snow is white’, the second—the token on the same line of ‘the second conjunct of the sentence on line 6 is not true.’ Thus, ‘the second conjunct of the sentence on line 6’ refers to the subtoken on line 6 that tokenizes this type. If α is ‘Snow is white’, we have:

$$p\downarrow = \alpha \wedge \neg Tr(p2)$$

$$p1\downarrow = \alpha \text{ and } p2\downarrow = \neg Tr(p2)$$

In both PTT and PTP, $p1$ gets **T** and $p2$, which forms a closed loop, gets *GAP*. But in PTT p gets *GAP*, while in PTP it gets **T**, via (SV). From PTP’s point of view, the token of the conjunction on line 6 and the subtoken that tokenizes the second conjunct are just two different pointers. The PTP system takes no cognizance of the fact that the second is a subtoken of the first. It is therefore not strange that the first gets **T**, while the second gets *GAP*; no more strange than the assignment of different values to tokens of the same sentence.

The PTP system can be modified so as to include operations that are given special status in the evaluation rules. The following variant has appealing aspects. Consider a negation-forming operation

$$neg()$$

which maps, in a one-to-one way, every pointer p to a pointer, $neg(p)$, pointing to the negation of what p points to:

$$neg(p)\downarrow = \neg(p\downarrow)$$

‘The negation of ...’ can be plausibly construed as referring to $neg()$; e.g., ‘The negation of the sentence on line 1’ is a name of $neg(p)$, where p is the sentence-token on line 1. Usually $neg(p)$ is not realized as a token. The exception is the case where p constitutes a subtoken of a bigger token of $\neg(p\downarrow)$; the bigger token can then serve as $neg(p)$. As a rule, $neg(p)$ is an additional posited pointer.

As far as the present system is concerned, $neg(p)$ is just another pointer to $\neg(p\downarrow)$. If p and $neg(p)$ get standard values, their values are opposites; but one may get *GAP*, while the other gets a standard value. For example, if $p\downarrow = \neg Tr(neg(p))$, then $neg(p)\downarrow = \neg\neg Tr(neg(p))$. Hence $neg(p)$ gets *GAP* via the closed loop rule and p gets **T**. This can be realized as ‘The negation of what I am saying now is not true’, or by writing on line 8 ‘The negation of the sentence on line 8 is not true.’

In order to treat $neg()$ in a way that reflects better the intuitive meaning of ‘the negation of...’, the procedure should be modified so that the value of $neg(p)$ derives from the value given previously to p . This is accomplished through changes, (I), (II) and (III), listed below.

Say that a pointer p is *syntactic* if it is of the form $neg(q)$; let us refer to q (which is uniquely determined by p , since $neg()$ is one-to-one) as p ’s *derived pointer* and denote it as $p1$. Now change the evaluation procedure as follows.

- (I) Restrict (SV), by requiring, as part of the enabling condition, that the pointer in question be non-syntactic.
- (II) Add a rule for evaluating syntactic pointers:
- (NEG) If $p = \text{neg}(q)$ and $v(q)$ is standard, assign to p the opposite value. If $v(q) = \text{GAP}$, assign to p GAP .
- (III) Change the direct calling relation. If p is not syntactic, the condition remains as before. If p is syntactic and unevaluated by v , then, by definition, p calls directly $p1$ under v .
The calling relation is defined as before in terms of direct calling.

The phrasings of the closed loop rule and the groundless pointers rule remain the same. Some concepts are adjusted in obvious ways. E.g., a valuation, v , is weakly self-supporting if for every evaluated p : (i) if p is not syntactic and $v(p)$ is standard, then $v(p) = \bar{v}(p\downarrow)$, (ii) if $p = \text{neg}(q)$, then q is evaluated and the value of p is obtained from it via (NEG). With these adjustments, all the main results hold.

Since $\text{neg}(p)$ calls directly only the pointer p , any closed loop containing the former contains the latter. It is possible that p belongs to a closed loop and $\text{neg}(p)$ does not, e.g., if $p\downarrow = \neg \text{Tr}(p)$. But then p gets GAP , and $\text{neg}(p)$ gets GAP via (NEG). It is also easily seen that p is groundless iff $\text{neg}(p)$ is. All in all, $\text{neg}(p)$ gets GAP iff p does.

Using $\text{neg}()$, we can reduce the number of primitive semantic predicates, via the equivalence:

$$Fa(p) \iff Tr(\text{neg}(p))$$

If we replace everywhere the left-hand side by the right-hand side and construe every pointer to the former as a pointer to the latter, the assignments of truth-values to pointers and to sentences, remain the same. The equivalence can therefore serve as a definition of $Fa(p)$.

We can also “save” on truth-values. According to the present rules, if one of p and $\text{neg}(p)$ gets **F**, the other gets **T**. We can allow, instead, an assignment of **F** both to p and to $\text{neg}(p)$, as a substitute for an assignment of GAP to p . The rules can be rephrased so as not to mention GAP . The change in the closed loop rule reads: For all p in the closed loop, assign **F** to p and to $\text{neg}(p)$. The groundless pointers rule is similarly adjusted. And the last clause of (NEG) is replaced by: If $v(q) = v(\text{neg}(q)) = \mathbf{F}$, assign **F** to $\text{neg}(\text{neg}(q))$. In the resulting system **F** signifies two kinds of failure: that of expressing a falsity and that of not expressing a proposition. The value of $\text{neg}(p)$ is used to distinguish between the two. The appearance of a two-valued “classical” logic is misleading, because $\text{neg}()$ is no longer truth-functional (the value of p does not determine the value of $\text{neg}(p)$). The gaps have been only disguised and they can be recovered: p is a gap iff both p and $\text{neg}(p)$ are false.

The above treatment of negation can be extended to other connectives. Consider, for instance, a conjunction operation that maps (in a one-to-one way) any two

pointers p, q to $con(p, q)$, which is a pointer to $p\downarrow \wedge q\downarrow$. The syntactic pointers now include also pointers of the form $con(p, q)$. If $r = con(p, q)$ then its derived pointers, $r1$ and $r2$, are p and q . Various adjustments are required, which I shall not go into. I hope that the main idea is clear.

The PTT system does not contain operations such as $neg()$ or $con(,)$. But each pointer, p , to a sentential compound or to a quantified sentence has derived pointers ($p1, p2, p|t$) to $p\downarrow$'s immediate components or to its instantiations. The assignment of values goes through these derived pointers. All pointers, except those that point to atomic sentences, become thereby syntactic and (SV) becomes a rule for assigning values to pointers that point to atomic sentences only.

5. FORMAL, PHYSICAL AND RANKED POINTERS

Consider an infinite sequence of rocks, each carrying the inscription 'What is written on the next rock is not true.'¹⁹ The inscriptions form an infinite descending chain, the same as (DC1) except that ' Tr ' is replaced by ' $\neg Tr$ '. All these pointers are groundless and we get a black hole. I doubt that we should take such physical situations seriously. There are nonetheless infinite descending chains that cannot be ignored.

Assume that \mathcal{L} contains first-order arithmetic. We can use Gödel numbers as pointers to sentences: $n\downarrow = \alpha$ iff n is a Gödel number of α . (We can insure, if needed, that every sentence has infinitely many Gödel numbers.) By well-known techniques we can define an infinite sequence: $n_0, n_1, \dots, n_i, \dots$, such that each n_i is a Gödel number of $\neg Tr(n_{i+1})$. Here we have to decree that each of these pointers fails to express the proposition to which it points, implying, among other things, that n_0 is not true, n_1 is not true, etc. But how can we say this without falling into the black hole? As a matter of fact we can and we do. I have just done so myself. But then I used an inscription, not a Gödel number. Suppose that a teacher, having set up the example on the board, points to n_1 declaring that it is not true (does not express a true proposition). Her utterance is a pointer, through which she succeeds in asserting $\neg Tr(n_1)$. Yet n_0 , which points to the same sentence, fails. The utterance reflects a point of view higher than that of n_0 ; it targets the whole arithmetical network of Gödel numbers. These considerations show that there is a place for an asymmetric treatment of the Gödel numbers on one hand, and the inscriptions and utterances on the other.

We can classify the Gödel numbers as formal (or, if you want, as mathematical or logical) pointers, and my inscription and the teacher's utterance as physical ones.²⁰ Physical pointers are, of course, formally represented in our system. But they mark external judgments of the agent, which relate to the whole formal part of the network. If we classify pointers in this way, then the groundless pointers rule should be modified: If there are groundless pointers, assign *GAP* to all unevaluated *formal* ones. We can then go on with the evaluation process. In our example this would result in the formal pointers (the Gödel numbers) getting *GAP* and the physical pointers getting **T**. Only when all groundless pointers are physical, do we assign to all of them *GAP*. A similar modification is suggested for the closed loop rule.

These reflections may help us to make sense of the infinite rock sequence. Assume that a superior point of view (say, that of some angel) takes in the full picture. A judgment ensuing from *that* perspective is to be represented by a higher-ranked pointer. *GAP* is first assigned to all the rock inscriptions; then the higher-ranked pointers to ‘The inscription on the i^{th} rock is not true’, $i = 1, 2, \dots$, get **T**.

This suggests an obvious generalization: Include in the pointer system a ranking function, that associates with every pointer, p , an ordinal number, $\rho(p)$, *the pointer’s rank*. Higher-rank pointers are supposed to represent, in cases of failure, a more comprehensive point of view; failure is first attributed only to pointers of minimal rank. We can let the ranks range over all ordinals smaller than some non-zero λ . Preferably, λ is a limit ordinal; this makes it possible to get a system satisfying the local metalevel requirement. In the sequel we assume, for simplicity, that $\lambda = \omega$; the ranks are thus natural numbers. The results remain valid if λ is any non-zero limit ordinal.

The standard-values rule (SV) remains the same. The revisability rule (SV’) has to be modified, but since this rule is not needed if \emptyset , or any “good” valuation, is the starting point, we can postpone its discussion. The modification of the other two rules is obvious. Whereas in the original rules *GAP* is assigned to all pointers in some set, we now assign it only to those pointers whose rank is the smallest among the set’s members.

Ranked Closed Loop Rule If X is a closed loop for v , assign *GAP* to all pointers of X of minimal rank.

Ranked Groundless Pointers Rule If there are groundless pointers under v , assign *GAP* to all groundless pointers of minimal rank.

Example: Let

$$p \downarrow = \neg Tr(q), \quad q \downarrow = \neg Tr(p)$$

Right at the beginning $\{p, q\}$ is a closed loop. If $\rho(p) = \rho(q)$, then p and q get *GAP*. Otherwise the pointer of smaller rank gets *GAP* and the other gets **T**. If, in addition:

$$r \downarrow = \neg Tr(p) \wedge \neg Tr(q)$$

then r , being external to the loop, gets a value after the application of the closed loop rule. The rank of r does not matter. If p and q are of equal rank, r gets **T**; if they are of different ranks, r gets **F** (because one of the conjuncts of $r \downarrow$ does).

Note: The ranks are relevant only in determining, in the cases of failure, which of the candidate pointers fail. In such situations higher rank pointers express judgments of higher level. But the ranks do not determine a metalevel hierarchy, because, as the last example shows, a lower rank pointer can successfully express the non-truth, or the failure, of a higher rank one. The graph-theoretic properties of the network always take precedence over the ranking function.

The main results of the previous section carry over fully to the ranked version. The definition of weakly self-supporting valuations is exactly the same. The rule (SV’), and the definition of strongly self-supporting valuations that is based on it, should be modified. Take again the last example: $p \downarrow = \neg Tr(p), q \downarrow = \neg Tr(p)$. If $\rho(p) < \rho(q)$, then p gets *GAP* and q gets **T**. Under (SV’) the assignment of *GAP* is revisable to an assignment of **F**, but this we do not want. We do allow the revision

only when it derives from pointers of smaller or equal rank. The adjustment is the rule (SV*) given below.

Let $v|k$, where k is a natural number, be the restriction of v to all pointers of rank $< k$. Recall that v^s is the standard part of v : its restriction to the pointers that get under it a standard value.

(SV*) If $p \downarrow = \alpha$, $\rho(p) = k$ and $\overline{v^s|(k+1)}(\alpha) = w$, then assign to p the value w .

There is another stronger version of this rule that allows revisions that derive also from assignments of GAP to pointers of smaller rank. In the above example, with $\rho(p) < \rho(q)$, if $v(p) = v(q) = GAP$, then v is not revisable under (SV*). But since $v(p) < v(q)$, we might want to force \mathbf{T} as the value of q . This is accomplished by:

(SV**) If $p \downarrow = \alpha$, $\rho(p) = k$ and $\overline{v^s|(k+1) \cup v|k}(\alpha) = w$, then assign to p the value w .

There are, accordingly, two variants that correspond to our previous strongly self-supporting valuations. Say that a valuation v (over ranked pointers) is *strongly self-supporting* if it is weakly self-supporting and no assignment of GAP by v is revisable via (SV*) in any extension of v . Define *firmly self-supporting* valuations in the same way, except that (SV**) replaces (SV*). Spelt out in full, the additional conditions for strong and firm self-support are:

(SSS) If $p \downarrow = \alpha$, $\rho(p) = k$ and $v(p) = GAP$, then, for all u such that $v \subseteq u$, $\overline{u^s|(k+1)}(\alpha)$ is undefined.

(FSS) If $p \downarrow = \alpha$, $\rho(p) = k$ and $v(p) = GAP$, then, for all u such that $v \subseteq u$, $\overline{u^s|(k+1) \cup u|k}(\alpha)$ is undefined.

The main results for the ranked case can be now stated simply by repeating the lemmas and Theorem 1 of sections 4. ‘Self-support’ can be interpreted throughout either as weak self-support, or as strong self-support, or as firm self-support. Theorem 2 holds if we add the assumption that the pointer in question, q , has rank \geq the ranks of the pointers it calls directly under \emptyset . Theorem 3 holds under its original assumptions (the ‘external’ pointer, q , succeeds even if its rank is smaller than the ranks of the pointers involved in α).

Theorem 4 If all pointers called directly by p , under \emptyset , have ranks $< \rho(p)$, then p does not fail.

Proof Similar to the proof of Theorem 2. If at some stage p gets GAP , it belongs at that stage to some X , which is either a closed loop or a groundless set. At least one of the pointers called directly by p , under \emptyset , must belong to X . Since its rank is smaller than $\rho(p)$, p is not of minimal rank among the members of X . But then it does not get GAP at that stage. Contradiction.

Corollary If p points to a sentential compound of $Tr(p_i)$ ’s and $Fa(p_i)$ ’s, $i = 1, \dots, n$ and its rank is higher than the ranks of the p_i ’s, then p does not fail.

Consequently, the local metalevel requirement is satisfied, provided that for every p there are pointers of higher rank to $Tr(p)$, to $Fa(p)$, to their negations, and to $\neg Tr(p) \wedge \neg Fa(p)$. This neutralizes the effect of groundless pointers and of large

closed loops. The general expressibility requirement is, however, not fully satisfied if \mathcal{L} is sufficiently expressive. The sentence of (L1) in section 4 does not have a non-failing pointer to it.²¹ It seems unlikely that the expressibility requirement can be fully satisfied for highly expressive languages. The problem touches on questions concerning the possibility of attaining a semantically closed language, or something near to it, which I shall not go into.

The ranked approach enhances considerably that which is stateable, without metalinguistic ascent, through the use of pointers. We pay for this by appealing to an additional primitive: the ranking function. Perhaps the price is not as high as it might look. The ranking can be derived from a partial ordering. Given any well-founded partial ordering, we can associate with it a plausible stratification of the domain—in our case the set of pointers—into ordinal ranks.²² It is therefore sufficient to stipulate certain sharp inequalities, of the form $p < q$ (meaning that p has smaller rank than q), which generate a well-founded partial ordering.²³ We can then use the associated ranking. If, for example, there are no inequalities, all pointers are of rank 0 and we get the non-ranked version.

In order that the local metalevel requirement be fully satisfied it is sufficient that, for each p , every sentential compound of $Tr(p)$ and $Fa(p)$ has a pointer, q , to it such that $q > p$ (i.e., $q > p$ is stipulated). This, and the well-foundedness of the generated partial ordering, are implied by the following two conditions.

- (C1) For each sentence α there is a pointer p_α to α , such that $p < p_\alpha$ for every p whose name occurs in α under a semantic predicate.
- (C2) There are no cycles: $q_0 > q_1 > \dots > q_0$, and no infinite descending chains: $q_0 > q_1 > \dots q_n > \dots$

(C1) implies that ' p_α ' does not occur in α under a semantic predicate. But if α contains quantifiers over pointers, then p_α is in their range. To restrict the quantifier ranges to pointers that are $< p_\alpha$ is to reduce the descriptive power of \mathcal{L} . We can, in this way, achieve the general expressibility requirement at a price of not being able to quantify over all pointers. As it is, we can achieve the local metalevel requirement, without reducing \mathcal{L} 's power. \mathcal{L} may even contain a relation symbol that denotes the partial ordering of the pointers.

Intuitively, p_α corresponds to a viewing of α in which α is fully laid out; a viewing that comprehends the pointers whose names appear in α as something already given. Formally, we can build the p_α 's into the setup by recursion. Given any \mathcal{L}_0 and a pointer system \mathcal{P}_0 for it, add for each sentence, α , a new pointer, p_α , pointing to α , and stipulate the inequalities $p_\alpha > p$ for every p whose name occurs in α under a semantic predicate. This yields an extension, say \mathcal{P}_1 , of \mathcal{P}_0 . Extend \mathcal{L}_0 to \mathcal{L}_1 by adding individual constants to name the new pointers. Extend \mathcal{P}_1 to \mathcal{P}_2 , by adding a new pointers, p_α —for each sentence, α , that is not in \mathcal{L}_0 —adding also the inequalities that go with them; extend \mathcal{L}_1 to \mathcal{L}_2 by adding constants to name the new pointers. And so on. The required setup consists of the language that is the union of the \mathcal{L}_i 's and the pointer system that is the union of the \mathcal{P}_i 's.

A better way of achieving (C1) and (C2) is to cast some special available pointers in the role of the p_α 's. Assume that, among other pointers, sentences function as pointers to themselves (cf. section 4 "Some Special Pointers"); we can then put: $p_\alpha = \alpha$, and posit $p < \alpha$ whenever p 's name occurs in α under a semantic predicate. This can be done, provided that the names of sentences-qua-pointers (i.e., under the semantic predicate) include the sentences as parts, as is the case when the names are formed by enquoting; because then (C2) is guaranteed, it is implied by the well-foundedness of the syntax. Other special pointers, such as the demonstrative pointers of the previous section, can play the role of the p_α 's; again, provided that their names contain, in this way or another, the sentences themselves as parts.

We have assumed here that the partial ordering is generated by the inequalities $p < p_\alpha$, where p 's name occurs in α under a semantic predicate. If, for whatever reason, there are other inequalities, then we must assume that their inclusion does not lead to a violation of (C2). The problem does not arise if we adopt the recursive construction above, by which the p_α 's are added as new pointers. The new inequalities can be joined to any well-founded partial ordering over the pointers of \mathcal{P}_0 . We can also choose the new pointers as sentences that point to themselves, provided that these do not figure already as pointers of \mathcal{P}_0 .

Other inequalities may be posited by virtue of the special nature of existing pointers. In the Gödel-number example, the inequality $p < q$ is posited whenever p is a formal pointer and q is an utterance or an inscription. We might also want to give higher rank to utterances made at later times, the idea being that they reflect a point of view that comprehends earlier utterances. If Jack says "What Jill will say is not true" and Jill, having heard Jack, says "What Jack said is not true", Jack's utterance gets *GAP* and Jill's utterance gets **T**. But there are no systematic guidelines for deciding, in general, which act of assertion is made from a more comprehensive stance. Here the pragmatic appeal to special circumstances cannot be avoided.

REFERENCES

- [1] BARWISE, J. AND ETCHEMENDY, J. 1987 *The Liar, an Essay in Truth and Circularity*, Oxford University Press.
- [2] BURGE, T. 1979 "Semantical Paradox", *The Journal of Philosophy* 76, pp. 169–198, reprinted in *Recent Essays on Truth and the Liar Paradox*, ed. Martin 1984, Oxford University Press.
- [3] DONNELAN, K. 1966 "Reference and Definite Descriptions", *Philosophical Review* 75.
- [4] DUMMETT, M. 1992 *The Logical Basis of Metaphysics*, Oxford University Press.
- [5] GAIFMAN, H. AND SHAPIRO, E. 1989 "Fully abstract compositional semantics for logic programs", Technical Summary, *POPL (Symposium on Principles of Programming Languages) January 1989*, ACM Press, 1989, pp. 134–142.
- [6] GAIFMAN, H. 1992 "Pointers to Truth" *The Journal of Philosophy* 89, pp. 223–262.
- [7] GUPTA, A. 1982 "Truth and Paradox" *Journal of Philosophical Logic* 11, pp. 1–60, reprinted in *Recent Essays on Truth and the Liar Paradox*, ed. Martin 1984 Oxford University Press.
- [8] HERZBERGER, H. 1982 "Notes on Naive Semantics" *Journal of Philosophical Logic* 11, pp. 61–102.
- [9] KRIPKE, S. 1975 "Outline of a Theory of Truth" *Journal of Philosophy* 72, pp. 690–716.
- [10] KAPLAN, D. 1989 "Demonstratives", in: *Themes from Kaplan*, ed. J. Almog, J. Perry and H. Wettstein, Oxford University Press.
- [11] KRIPKE, S. 1977 "Speaker's Reference and Semantic Reference", in: *Contemporary Perspectives in the Philosophy of Language*, ed. P. French, T. Uehling Jr. and K. Wettstein, University of Minnesota Press.
- [12] LEWIS, D. 1972 "Languages and Language", in: *Language, Mind and Knowledge*, ed. K. Gunderson, University of Minnesota Press 1975. In Martinich.
- [13] PARSONS, C. 1974, "The Liar Paradox" *Journal of Philosophical Logic* 3, pp. 381–412, reprinted in *Recent Essays on Truth and the Liar Paradox* ed. Martin 1984, Oxford University Press.
- [14] PRIOR, A. 1961 "On a Family of Paradoxes" *Notre Dame Journal of Formal Logic* II, 1, pp. 16–32.
- [15] RUSSELL, B. 1908 "Mathematical Logic as Based on the Theory of Types", *American Journal of Mathematics*, reprinted in *Logic and Knowledge, Essays 1901–1950*, R. Marsh ed. Macmillan Company 1956, pp. 59–102
- [16] SCOTT, D. 1970 *Outline of a Mathematical Theory of Computation*, Technical Monograph PRG-2, Oxford University Computing Laboratory Research Group.
- [17] SCOTT, D. AND STRACHEY, C. 1971 *Towards a Mathematical Semantics for Computer Languages*, Technical Monograph PRG-6, Oxford University Computing Laboratory Research Group.
- [18] STALNAKER, R. 1972 "Pragmatics", in: *Semantics of Natural Language*, ed. Harman and Davidson, Reidel 1972.
- [19] VAN FRASSEN, B. 1968 "Presumption, Implication, and Self-Reference", *The Journal of Philosophy* 65 pp. 136–152.

NOTES

¹ Even had we decided, for whatever reason, to regard the Truth Teller as true, we still would have had to distinguish between a statement based on a straightforward implementation of (*) and a statement that reflects a higher level view.

² A detailed argument for the impossibility of construing the two-line puzzle as a case of indexicality is given in PTT, pp. 246–248.

³ Thus, Dummett [1992] p. 31: "Every [Dummett's emphasis] semantic theory has as its goal an account of the way in which a sentence is determined as true, when it is true, in accordance with its composition."

⁴ An important condition, called *full abstraction*, prevents trivial choices of semantic values. In the case of sentences the condition requires that if two phrases of the same syntactic category have

different semantic values, then there should be a sentence whose truth-value is affected by substituting one phrase by the other.

⁵The point of using this or that semantics is the insight thereby gained into the phenomena in question. The semantic values underlying a denotational semantics should provide a good conceptual handle on what goes on. The satisfaction of certain formal conditions is, by itself, not sufficient. Various operational semantics can be converted into denotational ones by choosing as semantic values certain highly abstract complex entities (higher order functions). Such choices, however, require justification.

⁶ Following Wittgenstein's line in the *Tractatus*, one might argue that nothing true is ever said by Liar sentences. Their use is explained through the distinction between saying and showing. I must say that I find the appeal to that distinction overworked. In any case, a Tractarian position amounts to a crippling view of logic, which gives up on any systematic metalinguistic account.

⁷ The oscillating values of the Liar result from repeated evaluations, where the truth (falsity) of the sentence at a given stage implies its falsity (truth) at the next stage. These ongoing rounds should not be viewed as ascending steps in the metalinguistic hierarchy. At the end the theoretician must take a stance from which the situation is described. Essentially there are two perspectives: that of the failure of the Liar sentence and that of the non-failure of the statements made by the theoretician.

⁸ Burge [1979] appeals to a vaguely outlined "pragmatic implicature". Appeals to pragmatic factors are also made in Parson's paper [1974]. The system of Barwise and Etchemendy [1987] is based on a formal modeling of propositions. No ambiguity there. But it ends by gesturing at vague pragmatic considerations when it comes to deciding which proposition (of the infinite number of candidates) is expressed by a given utterance or inscription.

⁹ The object referred to by the speaker may not be the object picked literally by the description the speaker uses. Thus, Donnellan's [1966] 'The man with the champagne glass in his hand' and many other examples. Cf. also Kripke [1977].

¹⁰ The rule does not prevent loops that involve several sentences that refer to each other through quantification. It can be extended so as to block such loops as well. But then it gives rise to highly variable changes in quantification ranges, which might be found objectionable. The matter needs further investigation.

¹¹ Graph theoretically, these "looped trees" can be characterized as directed graphs, such that if there is a path from a to b and from a' to b , then there is either a path from a to a' or a path from a' to a (or both). Moreover, different nodes can have the same label, but on every path a label can occur no more than once.

¹² As a rule, the distinction between the unable-to-say paradox and the Strengthened Liar is not clearly made; 'Strengthened Liar' serves for both. I myself used a somewhat unclear terminology in [1992], referring to the unable-to-say paradox by 'Strong Liar'.

¹³ The immediate component of $\neg\beta$, is β ; the immediate components of $\beta \star \gamma$, where \star is a binary connective, are β and γ .

¹⁴ We can relax this rule a bit. Call a groundless set *connected* if it is not a disjoint union of two groundless sets. Call it a *groundless component* if it is connected and is not a proper subset of any connected groundless set. Every groundless set is a disjoint union of groundless components. In the relaxed version we assign *GAP* to all members of a groundless component. We therefore have freedom to handle each groundless component separately. This makes no difference, the final valuations are the same.

¹⁵ And so are the groundless component sets, if we use the more relaxed version of the groundless pointers rule, cf. footnote 14.

¹⁶ $\neg Tr(p) \wedge \neg Fa(p)$ is the simplest way of saying that p fails. In every reasonable system the existence of a non-failing pointer to that sentence implies the existence of non-failing pointers to its components. But since this was not assumed in the general setup, I have added, for good measure, the components. In view of Theorem 2, the requirement is of interest only if p fails.

¹⁷ Stated in terms of α , the condition is: α is unevaluated by $\bar{\emptyset}$ and there is a sequence $\alpha_0, \dots, \alpha_n$ of sentences unevaluated by $\bar{\emptyset}$ such that: (i) $\alpha_0 = \alpha$, (ii) every α_{i+1} is either an immediate sentential

component of α_i or an instantiation of it, and (iii) α_n is $Tr(q)$ or $Fa(q)$, where either $q = p$ or q calls p under \emptyset .

¹⁸Suppose p is groundless. Then, at some stage in the procedure, $\{p\} \cup X$, is a groundless set, where X consists of all pointers called by p , under the valuation at that stage. It is not difficult to show that if $q \downarrow = p \downarrow$ then, at the same stage, $\{q\} \cup X$ is groundless.

¹⁹I owe Gupta this illustration.

²⁰If ' p ' denotes the teacher's utterance, then $Tr(p)$ and its negation are sentences in \mathcal{L} . The Gödel numbering is not meant to include such sentences. If it does, we should distinguish between Gödel numbers of sentences that involve physical pointers and those that do not. Only the latter qualify as formal pointers. Note that any infinite descending chain (such as (DC1)) must either include an infinite descending chain of formal pointers, with a finite number of non-formal pointers tacked on, or it must derive from infinitely many independent pointings of physical pointers, resembling in this way the sequence of rocks.

²¹That sentence, namely $p \downarrow$, involves all the pointers that point to it. All of them belong to the same closed loop. Let $q \downarrow = p \downarrow$. Assume, for an inductive argument, that at the stage at which q gets its value every pointer, r , to $p \downarrow$, of rank $< \rho(q)$ has the value GAP . For these r 's the conjuncts $r \downarrow = p \downarrow \rightarrow \neg Tr(r)$ get **T**. But the conjunct $q \downarrow = p \downarrow \rightarrow \neg Tr(q)$ is still unevaluated, as are all conjuncts in which the pointer is of rank $\geq k$. Hence q , being now a pointer of minimal rank in a closed loop, gets GAP .

²²Let $<$ be an irreflexive partial ordering. A member $x \in X$ is a minimal member of X if there is no $y \in X$ such that $y < x$. The well-foundedness of $<$ means that every non-empty set has minimal members. Assign the rank 0 to pointers that are minimal in the set of all pointers. Assign 1 to the minimal pointers in the set obtained by removing all pointers of rank 0, and so on. In general, assign δ to the minimal pointers of the set obtained by removing all pointers of rank $< \delta$.

²³The partial order is the transitive closure of the relation defined by the set of these inequalities.

DEPARTMENT OF PHILOSOPHY, COLUMBIA UNIVERSITY, NEW YORK, NY 10027, USA
E-mail address: hg17@columbia.edu