A Solution to the Liar Paradox: Gaifman's Pointer Semantics

Authors: Chun-An Chou, Wen-Fang Wang

Abstract: It is well-known that some liar-like sentences lead to paradoxes. A sentence like

(1) (1) is not true.

leads to a paradox and thus is not true. So we conclude that "(1) is not true." Since we are repeating the very sentence, it seems that our conclusion is not true as well. But we think our conclusion is true.

To solve the aforementioned puzzle, Gaifman[1, 2] proposes Pointer Semantics. According to Gaifman, we should assign truth values to sentence tokens, and tokens are assigned truth values by an algorithm called "Pointer Semantics." Two tokens of the same sentence type might have different truth values. The liar sentence and the sentence we concluded, which are two tokens, are assigned the value GAP and true respectively.

Three points related to Pointer Semantics are considered in this paper. One is that the connective "→" should be interpreted by strong Kleene's 3-valued logic. Another is that some semantic predicates, e.g. "is gappy", which might appear in natural language, can be added to the language we have just considered. A third is that it is plausible to construe sentence tokens as a kind of truth bearers.

Introduction

The following sentence is a liar, sometimes called "the strengthened liar"[3], which claims the sentence itself untrue:

(1) (1) is not true.

If we assume that (1) is true, then (1) is not true. And if we assume that (1) is not true, then (1) is true. Since either way leads to contradiction, (1) is paradoxical or gappy. A gappy sentence is not true. So we write down that (1) is not true:

(2) (1) is not true.

We find that (1) and (2) are the same sentence, i.e. "(1) is not true." So it seems that (2) is the same as (1). But we also think that (2) is true for the reason that (2) states a truth. Thus there is a puzzle here.

The way Gaifman proposes to solve the preceding paradox is an algorithm called "Pointer Semantics". According to Gaifman, we should assign truth values not to sentence types, but to their tokens. The truth value of a token depends on not only its sentence type, but also the network among tokens. It can be shown that, from the network, (1) is self-referential, but (2) is not. The details can be explicated by the algorithm. The algorithm consists of three groups of rules: Standard rules, Gap rules and the Jump rule. A pointer, anything that points to something, is assigned a value by the algorithm. A token is a pointer which points to its sentence type. Two tokens pointing to the same sentence type might have different truth values. A token which fails to achieve a classical value is assigned by the Gap rules the value GAP. And tokens which do not "fail" are assigned, by the Standard rules or the Jump rule, the value T or F. (1) fails to achieve a classical value and is assigned GAP, but (2) is assigned the value T.

The formalism[1] consists of a language, a pointer system, a model, and the algorithm. A language L consists of individual constants, predicates, connectives, pointer-constants, and two semantic predicates T(x) and F(x). A pointer-constant, which is an individual constant, is interpreted as a pointer which points to a wff of L. For simplicity, we assume pointer-constants "p", "q", "r" … are interpreted as pointers p, q, r …, respectively. Two semantic predicates T(x) and F(x) take pointer-constants as argument. T(p) and F(p) are atomic semantic wffs.

A pointer system S, for a language L, is that:

PS = (P, $\downarrow$, ()1, ()2)

where P is a set of pointers; $\downarrow$ is an onto mapping from pointers to wffs of L; ()1 and ()2 are two functions associating pointers with pointers. The wff which p points to is called "p$\downarrow$." The two functions, ()1 and ()2, associate a pointer p with pointers p1 and p2 such that: (a) if p$\downarrow$= A*B (* is a binary connective), then p1$\downarrow$= A and p2$\downarrow$= B; and (b) if p$\downarrow$= ~A, then p1$\downarrow$= p2$\downarrow$= A; and (c) otherwise, p1$\downarrow$= p2$\downarrow$= p$\downarrow$.

A model M for L is that:

M = (τ , PS, δ)

where τ is a function which assigns every basic sentence (atomic sentence without semantic predicates) a value, either T or F; PS is a pointer system; δ is a mapping which associates every pointer-constant with a pointer.

Assume a given model M for L. A network among pointers consists of a pointer and all pointers it calls. We say that a pointer p calls a pointer q directly if (a) p$\downarrow$= ~A or p$\downarrow$= A * B, and either q = p1 or q = p2; or (b) p$\downarrow$= T(q) or p$\downarrow$= F(q). A calling path from p to q is a sequence of pointers $p_1, \ldots , p_n$, in which $p_1 = p$, $p_n = q$, and every p_i

calls p_{i+1} directly. p calls q if there is a calling path from p to q. Furthermore, a set of pointers S is a closed loop if S satisfies two conditions: (a) for all q $\in$ S, if q is called directly by p, then p $\in$ S; and (b) for all p, q $\in$ S, there is a calling path from p to q which is in S.

The algorithm has the form that:

If C(v, p) then v(p) := value

where C(v, p) is a condition on the evaluation v and the pointer p; and value $\in$ {T, F, GAP}. "If C(v, p) then v(p) := value" means if C(v, p) is satisfied then make the assignment v(p) = value. T and F are called standard values and we put that $-T=_{df}F$, $-F=_{df}T$.

The three groups of rules of the algorithm are the Standard rules, the Gap rules, and the Jump rule. The Standard rules are (a) if p$\downarrow$= A, and A is basic, then v(p) := τ (A); (b) if p$\downarrow$= ~A, and v(p1) is standard (i.e. T or F), then v(p) := -v(p1); (c) if p$\downarrow$= A v B then (i) if v(p1) = T or v(p2) = T, then v(p) := T; (ii) if v(p1) = v(p2) = F, then v(p) := F; (d) if p$\downarrow$= T(q) or p$\downarrow$= F(q) then (i) if p$\downarrow$= T(q), and q is standard, then v(p) := v(q); (ii) if p$\downarrow$= F(q), and q is standard, then v(p) := -v(q). The Jump rule is that if p$\downarrow$= T(q) or p$\downarrow$= F(q), and q is already assigned GAP, and p$\neq$q, then v(p) := F.

The Gap rules are consists of the Simple-Gap rule, the Closed-loop rule, and the Give-up rule. The Simple rule is that if all pointers called directly by p are assigned values, and none of the preceding rules applied to p, then v(p) := GAP. The Closed-loop rule is that if a set of pointers S is a closed loop, and none of the preceding rules applied to any p in S, then for all p, v(p) := GAP. The Give-up rule is that if S is not empty, and none of the preceding rules applied to any p in S, then for all p, v(p) := GAP.

Some consequences are implied by Pointer Semantics. One of them is that, as Gaifman says, Pointer Semantics implies strong Kleene's 3-valued truth table. A sentence has the form "A $\rightarrow$ A" might not be true. For example,

(3) If (3) is true, then (3) is true.

where "if … then …" is interpreted by strong Kleene's 3-valued logic. The token (3) forms a closed loop and is assigned the value GAP. And other sentence tokens saying "If (3) is true, then (3) is true" get the value T.

Some semantic predicates, e.g. "is gappy", are not predicates in the language we have considered above. It seems that there are many predicates similar to "is gappy", e.g. "undefined", "meaningless", "neither true nor false", which are in natural language. Thus, we might add the predicate “is gappy” to the language. Consider the following sentence which is sometimes called “the extended liar”:

(4) (4) is false or (4) is gappy.

The token (4) is assigned, by the Closed-loop rule, the value GAP. Other tokens saying “(4) is false or (4) is gappy” get the value T, since the second conjunct is assigned T. The Jump rule is modified for $p{\downarrow}=$ GAP(q): if $p{\downarrow}=$ GAP(q), and q is already assigned GAP, and $p \neq q$, then v(p) := T.

I think that it is plausible to construe sentence tokens as a kind of truth bearers. An objection to sentence tokens, by Gupta[4], is that: “A more nominalistic construal of truth bearers, such as their identification with sentence-tokens, is not plausible because of the obvious difficulty that there are unexpressed truths, just as there are many unnamed objects.” Supposing that what Gupta means is that there are facts for each truth and there are unexpressed facts for some unexpressed truths[5], it is not obvious that this is an objection to sentence tokens as truth bearers. It seems that if there are truths without bearers, as there are objects without names, then our theory of truth leaves out some truths. But I do not think that it is a defect for a theory of truth if it does not list all truths. The goal of a theory of truth is not to list all truths, but to identify the conditions of all truths. If the conditions are identified, then the conditions of all unexpressed truths are identified as well. Thus I do not think that the unexpressed truths lead to obvious difficulty. There is no obvious reason to reject sentence tokens as a kind of truth bearers.

References

1. Gaifman, H. 1988. Operational Pointer Semantics: Solution to Self-referential Puzzles I. In M. Vardi, editor, *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge, Pacific Grove, CA, March 1988,* pages 43-59, Morgan Kaufmann.
2. Gaifman, H. 1992. Pointers to Truth. *Journal of philosophy,* 89:223-261.
3. van Fraassen, B. 1968. Presumption, Implication, and Self-reference. *Journal of philosophy,* 65:136-152.
4. Gupta, A. 1982. Truth and Paradox. *Journal of philosophical logic,* 11:1-60.
5. Kirkham, R. L. 1995. *Theories of Truth: A Critical Introduction*. Cambridge: MIT Press.