Notre Dame Journal of Formal Logic
Volume XIX, Number 1, January 1978
NDJFAM

THE INTENSIONALITY OF THE PREDICATE '___ IS RECURSIVE'

CHARLES F. KIELKOPF

G. Lee Bowie's "An Argument Against Church's Thesis" provides an opportunity to clarify some points about our concepts of recursive and computable. In [1], G. Lee Bowie argued that there is a noncomputable recursive function and that there are computable functions which are very likely nonrecursive. Against his first claim, I shall argue, primarily, that Bowie has instead exposed the intensional character of the predicate '___ is recursive.' Against his second claim, I shall argue that he has used an eccentric sense of 'computable.'

Bowie does not want to identify functions with rules of correspondence or descriptions of correspondences so he uses Church's λ-operator. So, before beginning my rebuttal, I will show how I will use this operator as well as settle some terminological points. A set S is a *non-negative integer correspondence*, or alternatively: a sequence, if S is a set of ordered pairs of non-negative integers $\langle x, y\rangle$ such that for every x there is exactly one y such that $\langle x, y\rangle$ is in S. Let us call non-negative integer correspondences simply 'correspondences,' and call these correspondences 'functions.' We can here identify functions with such correspondences because we will restrict our attention to function descriptions of one argument place whose domain is the non-negative integers and whose range is a subset of the non-negative integers. By regarding functions as sets such as these correspondences we follow usual mathematical practice and Church in section 03 of [2] where he writes on p. 16: "In other words, we take the word 'function' to mean what may otherwise be called a *function in extension.*" However, some authors such as Church himself in [3] and Yasuhara in [4] do not identify functions with extensions such as correspondences; they identify functions with rules for generating correspondences. When he identified functions with rules, Church wrote on p. 3 of [3]: "we shall say we are dealing with *functions in intension.*" Although we will not deal with functions in intension, we will talk of the intensions of functions.

The λ-operator provides notation for presenting an intension of a

Received September 5, 1973

function as well as providing terms for denoting or referring to functions. Let us use expressions with one free variable to express intensions of functions. These expressions will officially be of the form: The non-negative integer $f(x)$ correlated with non-negative integer x by following procedure P. These expressions give the form of a term for denoting values of a function as well as present a rule for generating, or trying to generate, a correspondence. Of course, these expressions are usually abbreviated to the $f(x)$ formula. For instance, x^2 abbreviates: The non-negative integer x^2 correlated with non-negative integer x by multiplying x by itself. These abbreviations are given because usually the $f(x)$ is a formula which conventionally represents or encodes the procedure P, as does x^2. These abbreviations are used in at least three ways. When x is replaced by a numeral k, $f(k)$ is used to denote a particular number. The formula $f(x)$ may be used to talk about some-value-or-other of a function, -the so-called ambiguous value. Also $f(x)$ may be used to present a rule for generating a correspondence. Two abbreviations we shall use are $\mathbf{Z}(x)$ for: The non-negative integer $\mathbf{Z}(x)$ correlated with non-negative integer x by correlating 0 with x, and $\mathbf{C1}(x)$ for: The non-negative integer $\mathbf{C1}(x)$ correlated with non-negative integer x by correlating 1 with x. A fourth use for these $f(x)$ expressions is to form terms for denoting functions. When $f(x)$ is one of these expressions, the term $\lambda x[f(x)]$ denotes, i.e., when used conventionally is used to refer to, the correspondence or function whose value at x is $f(x)$. Thus $\lambda x[\mathbf{Z}(x)]$ denotes the function whose value is always 0 and $\lambda x[\mathbf{C1}(x)]$ the function whose value is always 1.

Bowie's example of an allegedly recursive but noncomputable function has as an intension: The non-negative integer $\mathbf{b}(x)$ correlated with x by making $\mathbf{b}(x)$ be 1 if R and $\mathbf{b}(x)$ be 0 if not-R. Here R will be some sentence, such as 'Caesar thought of the moon at the moment of his death,' whose truth value we cannot determine. Let us abbreviate the expression of this intension to $\mathbf{b}(x)$. Thus $\lambda x[\mathbf{b}(x)]$ denotes Bowie's allegedly recursive but noncomputable function. In this paper, I concede that $\lambda x[\mathbf{b}(x)]$ denotes a correspondence. I shall not investigate a suggestion that the definite description $\lambda x[\mathbf{b}(x)]$ can be used only attributively and not referentially in Donnellan's sense of these terms. See [5]. The suggestion would be that we cannot use $\lambda x[\mathbf{b}(x)]$ referentially because we cannot use it to pick out from $\lambda x[\mathbf{Z}(x)]$ and $\lambda x[\mathbf{C1}(x)]$ which one we are talking about.

Let me present a reconstruction of Bowie's argument that $\lambda x[\mathbf{b}(x)]$ is recursive but not computable. He assumes (1) through (4) to be uncontroversial.

(1) $\lambda x[\mathbf{C1}(x)]$ is recursive.
(2) $\lambda x[\mathbf{Z}(x)]$ is recursive.
(3) $\lambda x[\mathbf{C1}(x)]$ is computable.
(4) $\lambda x[\mathbf{Z}(x)]$ is computable.

He has us observe that we cannot use $\mathbf{b}(x)$ *qua* rule to compute the correspondence, whatever it may be, denoted by $\lambda x[\mathbf{b}(x)]$. He expresses this observation, misleadingly I think, as (5).

(5) $\lambda x[\mathbf{b}(x)]$ is not computable.

By accepting the law of excluded middle for *R* he concludes (6).

(6) $\lambda x[\mathbf{b}(x)] = \lambda x[\mathbf{C1}(x)]$ or $\lambda x[\mathbf{b}(x)] = \lambda x[\mathbf{Z}(x)]$.

From (1), (2), and (6) he concludes (7).

(7) $\lambda x[\mathbf{b}(x)]$ is recursive.

He does not use (3), (4), and (6) to show that (1) through (6) are inconsistent. Instead he concludes that (1) through (6) show '___ is computable' is an intensional context because sentences formed from it can change truth value upon substitution of co-referential terms. But notice that in (3), (4), and (5) '___ is computable' stands for a property of sets.

My rebuttal will be arguments that if 'recursive' is used in its primary sense (1), (2), and (7) should be written as the alternatives below and that (7a) is false. Then I shall argue that when 'recursive' is used in a secondary sense to stand for a property of functions, then, although (1), (2), and (7) are true, *it is reasonable to interpret* (5) *as false. In its primary* sense 'recursive' is used to stand for a property of abbreviations of expressions which occur within the square brackets of $\lambda x[\ ]$ terms, i.e., $f(x)$ forms. When used this way, 'recursive' stands for an intensional property of functions in the sense that it stands for a property of expressions of intensions of functions. When 'recursive' is used this way (1), (2), and (7) should be as in the a-versions below, where you may use 'function form,' 'ambigious value denoter,' or 'rule' instead of 'expression.'

(1a) $\mathbf{C1}(x)$ is a recursive expression.
(2a) $\mathbf{Z}(x)$ is a recursive expression.
(7a) $\mathbf{b}(x)$ is a recursive expression.

Some reasons for holding that 'recursive' is primarily applied to function forms are (A), (B), and (C) below.

(A) A standard way to define 'recursive function' is to give a syntactical definition. A function is said to be recursive if and only if it is one of the specified initial forms or is obtained from initial forms and previously defined forms by use of some specified definition schemata. See pp. 220 and 279 of [6] and p. 121 of [7]. Except for those who identify functions with forms of functions *qua* rules, this syntactical definition should properly be regarded as a definition of 'recursive function form,' because it is forms, and not sets, which have syntactical properties. (B) If we are asked to show that a function is recursive, we are, in effect, asked to show that a function form is recursive. We are presented with forms as on pp. 222-223 of [6] and pp. 122-123 of [7]. We are certainly not presented with a denumerable list of ordered pairs and asked to show that it is recursive. Remember we are identifying functions with correspondences. (C) Arguments that there are only denumerably many recursive functions proceed by showing that there are only denumerably many recursive function forms.

See Kleene's [6] sections **56**, **57**, and especially p. 283. This indicates that 'recursive' is applied primarily to forms. If it applied primarily to correspondences there should be some procedure for inspecting the $2^{\aleph_0}$ correspondences and selecting out the denumerable subset of the recursive ones.

For these reasons my inclination is to refrain from applying 'recursive' or 'nonrecursive' to functions *qua* correspondences, just as we apply 'Turing program' only to forms and never to correspondences. I will admit, though, that in a secondary sense of 'recursive' correspondences may be called recursive. But first let us note that we have blocked Bowie's alleged counterexample to the half of Church's thesis that runs: If it is recursive it is computable. In this statement of Church's thesis the 'it' in the antecedent must refer to the same thing as the 'it' in the consequent. Thus, even if both (7a) and (5) were true we would not have a counterexample because (7a) says a form is recursive while (5) says that a correspondence is not computable. Furthermore, we still do not get a counterexample if we rewrite (5) as (5a) and establish (5a) with Bowie's observation that $\mathbf{b}(x)$ is not computable because if it were we could, contrary to our assumptions, establish the truth of R.

(5a) $\mathbf{b}(x)$ is not a computable expression.

'Computable expression' will be precisely defined below. For the moment our interest is with the falsity of (7a). We would not get a counterexample to: If it is recursive it is computable, from (7a) and (5a) because (7a) is false.

Why do I say that (7a) is false, i.e., that $\mathbf{b}(x)$ is not a recursive expression? The fact that I cannot conceive of how to start to give a proof of the recursiveness of $\mathbf{b}(x)$ persuades me that $\mathbf{b}(x)$ is not a recursive form. Also I can give the following argument. Bowie has not argued that there is a recursively defined function form $f(x)$ which is noncomputable, where 'noncomputable' would mean that for some numeral k we cannot determine in finitely many steps, following the procedure encoded in $f(x)$, the numeral $f(x)$. So, without begging the question of whether or not there are noncomputable recursive correspondences, I can use the inuitive principle that all recursive function forms are computable in the sense that for any numeral k we can in a finite number of steps determine which numeral $f(k)$ is. With this intuitive principle, I can use Bowie's observation that $\mathbf{b}(x)$ is not computable to conclude that $\mathbf{b}(x)$ is not a recursive function form, i.e., not a recursive expression.

In a secondary sense, 'recursive' can be applied to functions *qua* correspondences. I submit that this secondary sense is given by D1. D1 is in agreement with Bowie's remark on p. 71: "it is recursive since there are recursive equations which generate it."

D1: $\lambda x[f(x)]$ is recursive $=_{df}$ There is an expression $g(x)$, $g(x)$ is a recursive expression, and $\lambda x[f(x)] = \lambda x[g(x)]$.

With D1 and (6), I have to accept (1), (2), and (7). But now do not I have a counterexample to: If it is recursive it is computable? Do not (5) and (7) give such a counterexample? No. Just as we distinguished the application of 'recursive' to expressions from its application to correspondences, we can distinguish the application of 'computable' to expressions from its application to correspondences. When 'computable' is applied to correspondences as in (3), (4), and (5) it is quite reasonable to believe that (5) is false. Bowie himself grants that there are these two uses of 'computable' by specifying in his (I) on p. 69, in effect, that $\lambda x[f(x)]$ is computable if and only if there is an algorithm A such that A computes $f(x)$ for appropriate input for x. Also ordinary use reveals that we would say that the formula x^2 is computable as well as saying that the sequence of squares is computable. I shall define these two senses of 'computable,' observe that under them (5) is false, and defend the definition of 'computable' for functions.

D2: Function form $f(x)$ is a computable expression $=_{df}$
There is a process P, an algorithm, such that given any numeral k use of P leads in a finite number of steps to production of numeral $f(k)$.

D2 is taken from Bowie's (III) of p. 69. For an elaboration of this idea of computable, see section **40** of S. C. Kleene's [8]. In the sense of D2, 'computable' stands for an intensional property in the sense that it applies to expressions of intensions. However, to get this result that '___ is computable' is an intensional predicate, there is no need to admit that there is any predicate of set terms for which the principle of substitutivity of co-referential terms fails. In D3 below, I use $(\exists u)$ because the values of the bound variable are sets and I want to suggest that in D3 the quantifier cannot be interpreted substitutionally as it can in D1 and D2.

D3: $\lambda x[f(x)]$ is computable $=_{df}$ $(\exists u)\{u = \lambda x[g(x)]$ and $\lambda x[f(x)] = \lambda x[g(x)]$, and $g(x)$ is a computable expression.$\}$

I think that D3 captures the intent of Kleene's stipulation on p. 228 of [8]: "if there is a computation procedure for a function, we call the function *computable*."

We have the trivial procedures of producing $\mathbf{C1}(k)$ as 1 for all k and $\mathbf{Z}(k)$ as 0 for all k. So we have (8) and (9).

(8) Function form $\mathbf{C1}(x)$ is a computable expression.
(9) Function form $\mathbf{Z}(x)$ is a computable expression.

From D3, (6), (8), and (9), we get not only (3) and (4) but also, and of more significance, (10), i.e., that (5) is false.

(10) $\lambda x\,[\mathbf{b}(x)]$ is computable.

Thus again we have blocked an alleged counterexample to: If it is recursive it is computable, if D2 and D3 are acceptable in principle even if not in detail.

D2 is difficult to defend because of vagueness about what processes P are legitimate. At the end of the paper, however, I give some negative conditions for legitimate computing processes. So I fear I have to let defense of D2 rest primarily on our intuitions of what is necessary and sufficient for a formula to be computable. D3 differs from Bowie's definition of 'computable' for functions. In my terminology, Bowie's (I) of p. 67 is: $\lambda x[f(x)]$ is computable if and only if $f(x)$ is a computable expression. Bowie's (I) blurs any distinction between computable expression, i.e., computable function form, and computable function. Instead, D3 presents an alternative definition of 'computable function' suggested by Bowie himself when he wrote on p. 72 from the point of view of an imaginary opponent: "Alternatively, put in your philosophical terminology, I can say that a function is computable *simpliciter* if it is computable *under some description.*" Bowie seems to concede that this alternative is the definition that a careful mathematician would give. Curiously, though, in the last paragraph of Part II of his paper, Bowie seems to reject this alternative, viz., D3, on the basis of the following weak argument. If Church's thesis is put as: "a function is recursive if and only if it is computable *under some description,*" Church's thesis can be refuted by the fact that it is highly probable that there is a nonrecursive function which is computable under some description. So, the argument goes, there is no point in defining 'computable function' as in D3. Of course, D3 could be the proper kind of definition even if it allowed counterexamples to Church's thesis. Also the question at issue here is about the more obvious half of Church's thesis, viz., can there be recursive noncomputable functions? So it is irrelevant to show someone who introduces D3 to block counterexamples to: If a function is recursive it is computable, that D3 does not block counterexamples to: If a function is computable it is recursive. Also I shall close this paper by arguing that Bowie has not shown that it is likely that there is a function computable in the sense of D3 which is nonrecursive. But let me note some positive merits of D3. First, D3 allows, as Bowie admits, a commonly made distinction between descriptions of a function, viz., $f(x)$ formulae, which provide an algorithm and those which do not. Second, D3 and D2 maintain for computability the distinction between a function *qua* correspondence and a form for the function. Third, D2 and D3 preserve and clarify Bowie's insight that in one sense '___ is computable' is an intensional predicate; but without having to grant that there is a predicate of set denoting terms which does not apply to co-referential terms.

Let me now try to dismiss Bowie's argument that there is a nonrecursive function computable in the sense of D3. Consider the expression: The non-negative integer $\mathbf{cf}(x)$ correlated with the non-negative integer x by making $\mathbf{cf}(x)$ be 1 if the x-th flip of coin C is heads and $\mathbf{cf}(x)$ be 0 if the x-th flip of coin C is tails. (The flips of the coin are counted and values of $\mathbf{cf}(x)$ are recorded to avoid getting two values for the same x.) Let us abbreviate this expression to $\mathbf{cf}(x)$, a form for the "coin-flip function." Bowie would argue that $\lambda x[\mathbf{cf}(x)]$ is computable because $\mathbf{cf}(x)$ is a computable expression by D2 but that it is highly improbable that $\lambda x\,[\mathbf{cf}(x)]$ is a recursive function in the sense of D1. He reminds us that there are $2^{\aleph_0}$ correspondences of non-negative integers into $\{0, 1\}$, i.e., 0, 1 sequences. He adds, in effect, that there are only denumerably many recursive expressions. So, at most, denumerably many of these 0, 1 sequences are denoted by terms of the form $\lambda x\,[g(x)]$ where $g(x)$ is a recursive expression. Then with the assumption that $\mathbf{cf}(x)$ is not a recursive expression, he concludes that it is highly improbable that there is some other recursive expression $g(x)$ such that $\lambda x\,[g(x)]$ denotes the 0, 1 sequences, selected at random from the $2^{\aleph_0}$ sequences of 0's and 1's by flipping coin C, denoted by $\lambda x\,[\mathbf{cf}(x)]$.

I will assume that $\mathbf{cf}(x)$ is not a recursive expression. Also I will not quarrel with Bowie's claim that the sequence, if any, denoted by $\lambda x\,[\mathbf{cf}(x)]$ is very likely not identical with any denoted by a $\lambda x\,[g(x)]$ with recursive $g(x)$. (Bowie defines a denumerable family of nonrecursive expressions such as $\mathbf{cf}(x)$ and thus increases the probability that some expression similar to $\mathbf{cf}(x)$ generates a 0, 1 sequence which is not generated by any recursive expression.) Instead, I will argue that it is doubtful that $\lambda x\,[\mathbf{cf}(x)]$ has a denotation and that the process encoded in the form $\mathbf{cf}(x)$ is not a genuine computing process. $\mathbf{cf}(x)$ is not really computable in the sense of D2. I should note that in the remainder of the paper I will be considering function forms primarily in their role as rules.

What 0, 1 sequence does $\lambda x\,[\mathbf{cf}(x)]$ denote? If it denoted at all, it would denote the sequence of 0's and 1's that would result by following $\mathbf{cf}(x)$ *qua* rule for denumerably many flips of coin C. Let us not raise here any anti-platonistic worries about the existence of nondenumerably many denumerable sequences. Let us worry only if $\lambda x\,[\mathbf{cf}(x)]$ can be used to refer successfully to any of them. If one believes that it is now fixed or determined how all possible flips of C will turn out, I grant that he can rationally believe that $\lambda x\,[\mathbf{cf}(x)]$ denotes that predetermined sequence for C, whatever it may be. However, if one does not accept this determinism, he has no reason for thinking there is such a sequence. Let me repeat that I am not saying that $\lambda x\,[\mathbf{cf}(x)]$ fails to denote because any of the $2^{\aleph_0}$ 0, 1 sequences fail to exist. Grant that they all exist. My claim is that $\lambda x\,[\mathbf{cf}(x)]$ fails to denote any one of them because its sense or intension, i.e., $\mathbf{cf}(x)$, can be used to denote only finitely many members of the range of any such sequence. Prior to the 100-th flip of C, $\mathbf{cf}(100)$ cannot be used to refer to any value of $\lambda x\,[\mathbf{cf}(x)]$. So we cannot say that $\lambda x\,[\mathbf{cf}(x)]$ is the correspondence such that for any x its value at x is $\mathbf{cf}(x)$. For all k greater than y, where y numbers the last flip of C, there simply is no $\mathbf{cf}(k)$. I want also to emphasize that I am not arguing that $\mathbf{cf}(x)$ makes $\lambda x\,[\mathbf{cf}(x)]$ fail to denote a sequence because $\mathbf{cf}(x)$ does not enable us to pick out that sequence. Just as I granted that $\lambda x\,[\mathbf{b}(x)]$ denoted a sequence, I would grant that λx [the number $h(x)$ such $h(x)$ is the value of Gauss's favorite function at x] may denote a function even if we cannot pick it out from this description. Gauss's favorite function could have been addition. I am arguing that we cannot make assertions such as: $\lambda x\,[\mathbf{cf}(x)]$, whatever it may be, is such and such, because prior to denumerably many flips of C there is no such sequence. I think that I have at least shown that it is dubious that $\lambda x[\mathbf{cf}(x)]$ can be used to refer to a 0, 1 sequence.

A difficulty in arguing that $\mathbf{cf}(x)$ is not a computing rule is that we admit that the rules for the constant functions such as $\mathbf{C1}(x)$ and $\mathbf{Z}(x)$ are computing rules. And following these constant function rules, viz., working on a denumerable list of the same numeral, hardly involves anything that could be called calculating. Hence, we cannot reject $\mathbf{cf}(x)$ as a computing rule because following it involves no calculation. Still, it is eccentric to call $\mathbf{cf}(x)$ a computing rule. Let us say that $\mathbf{Z}(x)$ meets the minimal conditions for being a computing rule. Admittedly, any listing of the values of $\lambda x[\mathbf{Z}(x)]$ depends upon physical conditions, but what the values of $\lambda x[\mathbf{Z}(x)]$ are does not depend upon any physical conditions. However, not only must certain physical conditions hold to list the values of $\lambda x[\mathbf{cf}(x)]$, but what the values of $\lambda x[\mathbf{cf}(x)]$ will be depends upon physical conditions which need not be as they will be. In other words, given that its sense is fixed by linguistic conventions $\lambda x[\mathbf{Z}(x)]$ has to have the values that it does have. But even when its sense is fixed $\lambda x[\mathbf{cf}(x)]$ does not need to have any of the values it happens to get. People at different times and places can list the values of $\lambda x[\mathbf{Z}(x)]$. But to list the values of $\lambda x[\mathbf{cf}(x)]$ one has to have access to a particular coin C, or in Bowie's full example: access to a particular machine. And there is no way of duplicating the coin or machine so that people can go elsewhere and at another time list values for $\lambda x[\mathbf{cf}(x)]$ and be guaranteed to get the same results as those using the original equipment. Also I would say that $\mathbf{cf}(x)$ does not really abbreviate a finite set of rules as Kleene on p. 227 of [8] says an algorithm should. I would say that $\mathbf{cf}(x)$ encodes a denumerable or indefinitely large set of rules. There is a separate rule for each flip of coin C. The reason we cannot take $\mathbf{cf}(x)$ and go off by ourselves and compute values of $\lambda x[\mathbf{cf}(x)]$ is that not enough rules are given in the formula $\mathbf{cf}(x)$. We need that particular coin C to continue to give us more and more rules. The formula $\mathbf{cf}(x)$ encodes the rule schema: Look at C for the indefinitely many more rules. The above are my reason for saying that the procedure for listing values of $\lambda x[\mathbf{cf}(x)]$ by use of $\mathbf{cf}(x)$ is not computing. Basically the reasons amount to saying that this listing of values is not computing because it is only the reporting of certain empirical observations.

But in closing I must add that Bowie's arguments have made a contribution. They force us to recognize that recursiveness and computability are primarily properties of expressions, i.e., intensional properties. Bowie's arguments help us to get clearer about what is not a computing process. To defend Church's thesis we have to say that a potentially denumerable list of 0's and 1's, or Yes's and No's, is not being made by a computing process if items are put on the list according to the outcome of an empirical event. For instance, if someone were following the rule: $\mathbf{S}(x)$ is 1 if x-th output from machine M is 1 and $\mathbf{S}(x)$ is 0 if x-th output is 0, that person would not be computing even if M were printing out 0's and 1's according to the rule: Print 1 if x is even and print 0 if x is odd.

REFERENCES

[1] Bowie, G. L., "An argument against Church's thesis," *Journal of Philosophy*, vol. 70 (1973), pp. 66-73.

[2] Church, A., *Introduction to Mathematical Logic I*, Princeton University Press, New Jersey (1956).

[3] Church, A., *The Calculi of Lambda-conversion*, pp. 1-3, Princeton University Press, New Jersey (1941).

[4] Yasuhara, A., *Recursive Function Theory and Logic*, Academic Press, New York (1971), pp. 66-69.

[5] Donnellan, K., "Reference and definite descriptions," *Philosophical Review*, vol. 75 (1966), pp. 281-304.

[6] Kleene, S. C., *Introduction to Metamathematics*, Van Nostrand, Princeton (1950).

[7] Mendelson, E., *Introduction to Mathematical Logic*, Van Nostrand, New York (1964).

[8] Kleene, S. C., *Mathematical Logic*, John Wiley & Sons, New York (1967).

The Ohio State University
Columbus, Ohio