MIT/LCS/TM -109

EFFECTIVENESS

Rohit Parikh

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LABORATORY FOR COMPUTER SCIENCE

CAMBRIDGE MASSACHUSETTS 02139

MIT/LCS/TM-109

EFFECTIVENESS

Rohit Parikh

July 1978

EFFECTIVENESS[1]

Rohit Parikh

July 29, 1978

Abstract: Church's thesis equates the intuitive notion 'effective' with the mathematical notion 'recursive'. In order for this thesis to provide any information to us we have to have a clear understanding of both notions. We consider one of the prevalent definitions of 'effective' and compare it with the notions of syntactic and semantic consequence to see which one it corresponds to better. The notion of syntactic consequence, while useful, is subservient to the semantic notion and when we go from one language to another we expect to have to change the syntactic notion of consequence, if we are lucky enough to have one at all. Similarly the prevalent notion of effectiveness is a restricted one and has had the effect of limiting our view. At the end of section 3, we give a more general analysis of effectiveness and propose a mathematical theory. In section 4 we consider the question whether the set of grammatical sentences of English is recursive. We show that this question is not well posed and that the arguments in favour of a positive answer are question begging. We reformulate this question in the form "How recursive is the set of grammatical sentences of English?", and propose a way of turning it into a precise technical problem. The method used is a generalisation of the Kolmogorov-Chaitin theory of randomness which is briefly sketched.

Key words: Effectiveness, Church's Thesis, Turing machines, Program Correctness, Grammar.

This research was partially supported by NSF grants MCS76-18461 and MCS77-19754-A02

The following definition[2] of effectiveness has come to be widely accepted in the discussions of Church's thesis in the literature:

Def 1: A task is effectively performable iff there exist explicit instructions for performing it.

I shall begin by discussing this definition and show that while this definition is very useful in the context of Turing machines and programs, it is nonetheless circular and therefore useless in more general contexts where we are investigating a *new* situation with a view to knowing what an appropriate formal definition of effectiveness in this context might be.

Let us consider a related situation in logic. It is conventional there to give two separate definitions for the notion of logical consequence. One of these definitions defines the notion of semantic consequence usually denoted $\models$. This notion is defined by reference to models, valuations, or whatever. (E.g. in the predicate calculus $\Gamma \models A$ means:"A holds in all models of Γ".) The other definition gives a syntactic notion of consequence, usually denoted $\vdash$. ($\Gamma \vdash A$ usually means: "There is a formal derivation of A from Γ.") This syntactic notion is more useful in practice in that it actually allows us to prove various theorems, though, perhaps, for independence results, it is the semantic notion that is more useful.

Be that as it may, there is a clear feeling that in some sense it is the semantic notion that is primary and the syntactic notion is subservient to it[3]. A soundness result establishes that $\vdash$ is included in $\models$, whereas a completeness result shows that $\models$ is included in $\vdash$. In each case, it is the soundness or completeness respectively of $\vdash$ that we are worried about, and not that of $\models$. As further evidence of the same point, note that different formalisations, say, of predicate logic, can differ quite radically in their manner of defining $\vdash$, but agree on their definitions of $\models$, at least for sentences.

The question that now arises is, does the definition of effectiveness that we gave above correspond to $\models$ or $\vdash$? This question really amounts to asking if that definition (let us call the notion defined by it 'effective-1' from now on) sets a

goal for us, as the semantic definition did for the notion of consequence, or if it gives us a proposed solution to an already understood goal, as the syntactic definition did for consequence[4]. If we call $\models$ a goal notion or a g-notion, and call $\vdash$ a method notion or an m-notion, then our question is: "Is 'effective-1' a g-notion or an m-notion ?"

This question is quite crucial for us for if 'effective-1' is an m-notion, then we can expect it to be quite useful in cases that are well understood but to be a very poor guide in cases that are, as yet, uncharted. And this is because knowing a more or less detailed *solution* in one context does not give us much guidance in a novel situation, whereas a clear understanding of our goal would at least define the problem clearly in a different context though it may not tell us how to go about solving it. Thus for instance, $\models$ can be easily generalised to second order logic by merely changing the class of models, but $\vdash$ has no obvious analogue. If we are to study more than one notion of effectiveness, then we have to be quite clear about what it is that ties them all together.

SECTION 2

Before we proceed any further with our inquiry, perhaps it is well if we settle a preliminary question. Doesn't Church's thesis, which equates effectiveness with recursiveness *solve* the problem of characterising effectiveness ? And if it does, then why should we care if a particular definition of effectiveness is like $\models$ or like $\vdash$?

Now, of course, Church's thesis is not accepted by everyone[5] without any reservations. And while it may be true that people with reservations about it are a minority, they form a somewhat larger fraction of people who *consider* the question at all. But even if we did accept Church's thesis, that would only tell us that the two notions of effectiveness and recursiveness were equal, and unless we had *already* a notion of effectiveness, we would not know just what it was that Church's thesis was telling us. It is useless to be told that $a = b$ if all that we know about a is that it is equal to b. To see this more concretely, consider the question whether we can write a program to translate mechanically from German to English. In order to decide if we should take on this task, we have to decide first if we regard the task as effectively performable. And here Church's thesis is of no help at all

unless we have *already* written such a program, and in that case, naturally, Church's thesis is redundant.

Thus Church's thesis, in itself, does not make our inquiry superfluous. But are there, in any case, situations where Church's thesis is not even indirectly relevant?

Surely there are many such situations. Shepherdson[6] mentions the problem of packing sardines though he seems not to be very optimistic about the possibility of an interesting theory to cover such cases. Engeler[7] regards the famous ruler compass problems as effectiveness problems. (See also the very interesting paper[8] by Luckham, Park and Paterson on program schemata. The operations which are used to construct their schemes do not have to be recursive, only effective, for the whole program to make sense as an effective procedure.) Surely a cooking recipe or directions for getting from one place to another are examples of algorithms and the tasks that they enable us to perform must be regarded as effectively performable under ordinary circumstances. In general, we may say that an advance in technology, or sometimes just an advance in our understanding of some area enables us to transfer a particular task from the domain of 'non-effective' (or, more precisely, 'not known to be effective') to the domain of 'effective'.

To be sure, when we consider tasks as general as these, we may have to allow the existence of several different notions of effectiveness and even various degrees of effectiveness. But the point is made that *some* tasks that do not consist of manipulating strings are thought to be effectively performable and some other tasks of the same nature are not[9]. If we want even the beginnings of a theory in this area (and surely any theory here would have some use) then we must wean ourselves away from the notion that Church's thesis solves the problem of effectiveness once and for all.

SECTION 3

To return now to the question we raised originally. Is 'effective-1' a g-notion or an m-notion ? If the former, then we can just *use* it as a general guide in our inquiry. If the latter then we should look for a better candidate which works more

generally and also ask what is special about Turing machines etc. that makes *Def. 1* work there.

We tend to think of a Turing machine[10] as an object. However, we may, if we like, also think of it as a program operating on structures. The sorts of structures on which it operates are doubly infinite binary strings (these are mostly blank in ordinary circumstances) with exactly one state symbol occurring somewhere. Each quadruple is an instruction, and the problem of scheduling is neatly solved by stipulating that for each structure, at most one of these instructions *can* apply. Thus if the structure looks like ...bcq$_1$cd... , then only a quadruple that begins with q$_1$c can apply and it is part of the definition of a Turing machine that there is at most one such quadruple.

If we put aside the problem of scheduling for a moment, then we have a finite number of atomic actions that are possible in such a context. These consist of writing a symbol, one of finitely many, moving right, moving left, and if you like, halting. For a particular computation that does halt, what is performed is simply a sequence of such atomic actions. Thus *given* the set of possible atomic actions, the problem of effectiveness reduces to the problem of scheduling, i.e. of determining how the particular sequence of atomic actions that is performed is going to depend on the particular structure or input. (In programs this job is often performed by tests.) This second problem is of course not trivial, but it is only in the context of a finite complete set of atomic actions that the solution to the scheduling problem yields a solution to the whole problem of effectiveness.

Consider now a recipe for scrambled eggs as an algorithm. It might go: "Break three eggs in a bowl. Add half a teaspoon salt, and a quarter cup milk, stir thoroughly, ... ". It can't be said that the various actions described are *atomic.* In the first place, someone who does not know English will not be able to perform them. That means that these instructions are not written in whatever would correspond to 'machine language' for people. Otherwise they would be the same for all humans. But apart from that, each step really consists of a sequence of substeps which are not mentioned, but have to be performed nonetheless.

Thus the various steps in our algorithm are not atomic but really look like subroutines. But in *what* language are these subroutines written ? It is not a language that any of us can speak or program in. These steps that turned out not to

be atomic are not created by programming but by training. One learns to break eggs by watching other people do it and by trial and error. One is not necessarily aware of the little steps that make up the process of breaking an egg. Riding a bicycle is perhaps an even more extreme example of this. Very few people who ride a bicycle have any idea of just what it is that they are doing in the sense of being conscious of all the little actions that are necessary to preserve stability. If this were not so, the problem of making a robot that can ride a bicycle would be easy, at least in theory, but in fact it is quite hard - the robots keep falling off.

Compare now the recipe for eggs with the following 'algorithm' for deciding the truth or otherwise of number-theoretic statements: "Write the given formula as a truth-functional combination of formulae that are either atomic or begin with a quantifier. Decide the simpler formulae thus obtained. Now use truth tables to decide the original formula." A more facetious algorithm solves the problem posed by the following question. "How do you put four elephants in a VW bug ?". Answer: "Two in the front. Two in the back." In each case there is no algorithm, but the only way to see this is by seeing that the individual steps are not effective. And we don't know yet *how* we do this. Until we have discovered this, *Def. 1* is circular in that it presupposes what we mean by the effectiveness of individual steps. And *after* we have done this, the problem that it solves for us is trivial.

This problem did not arise with Turing machines because we had an exhaustive list of effective atomic actions, and any other action was acceptable as effective only if it could be reduced to these, but in the present context we have no such list. Still we know that the 'algorithms' for seating elephants and for deciding number-theoretic truth are not genuine algorithms, but the recipe for scrambled eggs is a genuine algorithm, albeit in a condensed form. How do we know ? The simple answer is that we know that under ordinary circumstances, people *can* carry out the individual steps. And of course, if people can carry out the individual steps then they can also carry out the whole algorithm, provided only that they can remember[11] it, and not get too tired on the way etc. Thus we have a second definition.

Def 2: A task is effectively performable iff it is possible for sufficiently many people, perhaps aided by some (fixed) instructions, to carry it out under ordinary circumstances.

This definition has a certain amount of vagueness. How many people have to levitate before we consider the task to be effectively performable ? On the other hand if we replace 'sufficiently many' by 'all' in the definition, then we get too restrictive a notion of effectiveness. No matter how simple a task, you can always find *some* people who are either unable or unwilling to do it! Nonetheless this definition does express what we intend the word 'effective' to mean and it does shed a certain amount of light on various problems. To take a particular example, it is often asserted in logic texts that Church's thesis cannot be proved because it asserts the equivalence of two notions of which one is intuitive whereas the other is mathematical However, Kreisel[12], has pointed out that such arguments are not really relevant.

Almost always, when one is attempting to give a mathematical definition of an intuitive notion, one attempts to put down sufficiently many intuitively evident properties of the intuitive notion. One then tries to prove that there is a *unique* mathematical notion that has these properties. A very good example of this is the Riemann integral . One writes down certain intuitive properties of the notion of area, e.g. that the area of a disjoint union of two figures is the sum of their areas. It can then be shown that at least for continuous functions, the Riemann integral is the unique functional having the stated properties.

In the present case, where we are considering the equivalence of 'recursive' with 'effective' (where we are considering only the string manipulation aspect of the second notion) we know enough properties of the intuitive notion that we can convince ourselves that

$$\text{recursive} \subseteq \text{effective}$$

provided we have no qualms about assuming unlimited time and space resources. However, even with this generous allowance, we have difficulty being equally convinced of the reverse inclusion

$$\text{effective} \subseteq \text{recursive}.$$

Or if we *are* convinced, it is not for logical reasons, but simply because no one has found any counterexamples yet. An informal *proof* of the second, problematic inclusion is lacking. And this problem is immediately traceable to *Def.* 2.

It has on the face of it, the property, that when the definition *is* satisfied, then at least in theory we can find out that it is. On the other hand if the definition is *not* satisfied then the definition gives no immediate help in discovering this fact. Thus we can see why it happens that the 'recursive $\subseteq$ effective' part of Church's thesis is less problematic than its converse. To establish the first, we merely have to convince ourselves that the various atomic actions that are part of the definition of 'recursive' are in fact effective. But to get the converse, we need some control or upper bound on the atomic actions[13] that people can perform. If we had a biological theory of people that told us what atomic actions people might be able to perform, then of course we would have such an upper bound. As things stand, this theory is lacking, and at least for some people, a belief in Church's thesis may be an indirect way of asserting that there is such a theory.

The analogy between 'effective-1' and $\vdash$ that we have been hinting at has a rather concrete part. An algorithm is rather like a proof with concatenation or ; (which stands roughly for: "and then do") as the sole rule of inference and the basic steps as the axioms. In defining the notion of recursiveness, the 'axioms' are explicitly given, and we have what amounts to a completely formalised system. In defining informal human effectiveness, the basic steps are still the 'axioms', but the criterion for their being so is no longer that they belong to some fixed finite list given in advance. Rather it is that we find them plausible. If we accept that each step in an algorithm is effectively performable, then we will accept the whole thing. But we no longer have any way of deciding *when* we will find each step of an algorithm so acceptable. Thus while 'effective-1' and and $\vdash$ are both m-notions, the latter is more exact and a better analogy would be between $\vdash$ and 'recursive' whereas 'effective-1' corresponds better with 'has an intuitively acceptable proof'. Finally, there is a third analogy, namely that between 'valid' and 'effective-2'. Unfortunately the latter notion lacks a mathematical correlate that would correspond to $\models$. Wanting some mathematical correlate, we took 'recursive', but that is an m-notion, not a g-notion and has had the effect of limiting our field of vision.

Before we finish with this section, one or two particular observations may be in order. During recent years a theory of program correctness[14] has evolved whose purpose (among others) is to prove that a program does what it is supposed to do. In this context we have assertions of the form $P \rightarrow \langle a \rangle Q$ which stand roughly for: "If the condition P holds when the program a is started, then the program terminates with

condition Q holding". (We are assuming for simplicity that the program is deterministic). Now it is easily shown that these correctness statements, as they are called, do not completely characterise a program. But in the theory of programs, we also have other ways of saying what a program is or does. Now when we condition or train animals or people, what we are doing is asking the subject to come up with a program which satisfies some given correctness condition. E.g. "when the bell rings(P), this lever should be depressed(Q)". We have no way of controlling what goes on in *between*. Moreover, apart from stipulating that condition Q is fulfilled at the end, we do not have much control over what things look like otherwise. (All of us can carry out the instruction "Write 'John Smith'", but the banking system depends on the fact that only John Smith himself carries it out in a particular way. To be sure, we can also find differences between the way two different computers ,or more precisely, printers, carry out this instruction, but the theory in that case *ignores* the differnces.) Thus for a procedure that people follow, the correctness conditions that it satisfies, are nearly *all* that we know about it. Of course, we can break up an algorithm into smaller bits, and impose conditions along the way, but there is a limit to how far one can go- there is no way to completely characterise a 'computation' as one can do with computers. Whereas in the context of computers, we start with *detailed* knowledge of the program and try to come up with its correctness conditions, so that if we are lucky, we have *both*.

Thus we see that structurally, there is a crucial difference between a computation and an ordinary procedure. What the latter is, is not a program but a sequence of correctness assertions. A procedure $(P_1,...,P_n)$ stands for: "Find effective ways $a_1,...,a_{n-1}$, such that for each $i < n$, $P_i \rightarrow \langle a_i \rangle P_{i+1}$ is true". If the ways $a_1,...,a_{n-1}$ are found, then we have a way of going from P_1 to P_n. In many ways these sequences *look* like programs. They can be composed by concatenation when the ends can be matched. If I have a sequence $(P_1,...,P_n)$ and another sequence $(Q_1,...,Q_m)$, then I can concatenate them if P_n implies Q_1, and create the new sequence $(P_1,...,P_n,Q_2,...,Q_m)$. We can then go on and build a theory of computation based on such sequences which will behave in some ways like recursive function theory. However, there are some significant differences. First of all we don't need to bring in atomic instructions at all. Secondly, we have a partial ordering here that becomes important. If we think of $(P_1,...P_n)$, as the set of all procedures that start with P_1 holding (input assumption) and achieve successively, conditions $P_2,...,P_n$, then, for instance, if Q implies R, then (P,Q) is included in

(P,R), whereas (R,P) is included in (Q,P). We also have the following laws that will hold quite generally.

1) $(P,Q\wedge R) = (P,Q)\cap(P,R)$ 2) $(P,Q)\cup(P,R) \subseteq (P,Q\vee R)$

3) $(P,R)\cup(Q,R) \subseteq (P\wedge Q,R)$ 4) $(P\vee Q,R) = (P,R)\cap(Q,R)$

Proceeding this way we can build a general mathematical theory of which the usual recursive function theory will be the atomic, finitely generated case, but which will also have atomless models. Some of these models may well be suited to analysing situations where effective procedures do not proceed in discrete time.

SECTION 4

Let us now consider the question of effectiveness in the more familiar context of strings where Church's thesis does have relevance. It has been argued by Hilary Putnam[15] that the set of grammatical sentences of a natural language is recursive. The argument he gives for this conclusion goes roughly as follows:

We may take it that the human brain may, for our purposes be thought of as a Turing machine. Since human beings can recognize[16] whether a given string is grammatical or not, the set of grammatical strings must be recursive.

Since such arguments are widely accepted, I would like to examine them here. I shall argue, not that Putnam's conclusion is incorrect, but rather that it is theory-laden. There is nothing wrong with the theory as such, no doubt it is correct, but since the answer to the question is already built into the theory, it does not give us the information we want. I shall start by developing my argument that Putnam's conclusion is built-in. Later in this section I shall sketch a technique for asking the question in such a way that the answer is not built in and can give us the information we want. The new form of the question could be formulated as: "how recursive is the set of grammatical sentences of a natural language?".

Let us now reconsider the argument that the set of grammatical sentences of a natural language is recursive. Let C_1 be the (finite) set of strings (of words or

letters) that we have admitted as grammatical. Similarly let U_1 be the set of strings that we have rejected as ungrammatical. And finally let G be the set of all grammatical strings. Then we have the following conditions:

1) $G_1 \subseteq G$ 2) $G \cap U_1 = \emptyset$

These conditions do not determine G. There are uncountably many solutions. Then why not simplify matters for ourselves by taking G to be equal to G_1? It would be recursive as well as finite.

The objection to this admittedly frivolous suggestion is that we know perfectly evident rules that force G to be infinite. An example would be " if x, y are grammatical, so is 'x and y' ", thus G cannot equal G_1.

Thus the principal reason why we consider the set of grammatical sentences to be recursive is that it is closed under certain rules. The obvious solution for G now is that G is the smallest set containing G_1 and closed under the rules. If there is any uncertainty about G, it is only because we are not sure we have all the necessary rules. And if we do not know any good argument for the criterion "smallest", we don't know anything against it either and taking G to be the smallest set, etc. does at least have the effect of determining G.

However we have now forced G to be r.e., and if we can get by with length increasing rules, or with rules that do not decrease length drastically, G will be recursive.

Clearly there is something suspicious about an argument that yields the r.e.-ness of G with so little trouble. We did not need to know anything specific about grammaticality at all. I shall now show that the argument isn't really sound in that it may yield the same sort of conclusion even in situations where the conclusion is false. I shall use two quite different examples to this end.

Consider the following *hypothetical* situation. Suppose that the human brain has the capacity to ask questions of some nonrecursive oracle. This oracle may be God or perhaps an inhabitant of some other universe connected to ours by your favorite scientific device. However, the oracle is not for G but for some other nonrecursive set[17]. We are ourselves not aware of this, but in fact we use this oracle to decide

questions about the grammaticality of English sentences. I am supposing, of course, that G is nonrecursive. The argument given above for the recursiveness of G is still going to work. Our data will still consist of two finite sets G_1, U_1, some finite set of rules, and we will still think G to be the smallest set containing G_1 and closed under the rules. Since we are unaware of the communication with the oracle we will require the rules to be recursive for theoretical reasons and will come out with the false conclusion that G is r.e. if not recursive.

Thus we have rigged our theory in favour of a positive answer to the question of the recursiveness of G. If we compare the situation here with that in number theory, we see that our rigging the answer there in the same way is prevented by the fact that we have an independent definition of the corresponding set, namely the set NT of true sentences of number theory. There we also have a finite set of known elements of the set, and a set of recursive rules under which we consider NT to be closed. The closure, say is Peano Arithmetic[18]. But having a *definition* of NT enables us to show that Peano Arithmetic is not NT.

The principal difference between the case of G and that of NT is that with NT we have an *independent* definition which we lack with G. But now we are looking for an answer to the question "Is G recursive?", to the same theory on which we depend to tell us what G is. That is the sense in which the answer is rigged.

To see this more clearly, let us consider another argument in favour of the recursiveness of G. This second argument explicitly rejects the oracle I mention above, presumably on the grounds that there is no evidence for it. It follows, then that since the human brain can effectively process questions about the grammaticality of sentences without any outside aid, the set of grammatical sentences must be recursive.

It is sought here to establish the conclusion that G is recursive, on empirical evidence which is somewhat doubtful. But even granting this evidence, there is an inherent risk in using empirical evidence in a situation as complex as that of recursive function theory. Suppose for example that the human brain uses an algorithm for G which depends for its correctness on a mathematical assumption which is false. E.g. suppose the algorithm depends on the assumption that the Burnside conjecture is true, when in fact it fails for large n. Then the algorithm may in fact work in practice, but does not establish the recursiveness of G. One could, of course take

the point of view that whatever this algorithm accepts should be called the set of grammatical sentences. However, this is again a circular argument of precisely the kind that I objected to earlier. Moreover, even this device will work only if everyone uses the same algorithm. We have no evidence for this beyond the fact that people accept (more or less) the strings in C_1 and reject the strings in U_1. And that evidence has already been rejected as insufficient. Any convincing evidence must now account for all of C and not just C_1, since it is precisely the part $C-C_1$ where the problems arise. If we do think of C as a set that has some independent characterisation, and we are not assuming at the very outset that it is r.e. then it would seem that the existence of algorithms of the sort I considered would be an evolutionary necessity. For there is no reason whatever to assume that the algorithm that works best *all* the time is always better than the one that works best for all the cases that arise in *practice*.

Thus what the empirical argument argument really establishes is the recursiveness not of C but of C_1, and that recursiveness need not be established since C_1 is finite. But now, consider the (finite) set of street names in Boston together with the binary predicate "intersects". The average cabby knows the diagram of this finite first order structure, but we are not tempted therefore to call it recursive or not recursive. We assume that the cabby has a large enough memory which he uses to answer questions about this structure, and that there are very few mathematically interesting properties that this structure has.

The street diagram of downtown Boston comes fairly close to being a random binary relation, as we all know. Why can't C_1 be equally random?

It isn't, of course, but we can see now that calling it recursive is entirely beside the point. What we really want to know is: "to what extent are questions about C_1 answered from *memory* and to what extent are they answered on the basis of some sort of *computation*?" In other words, *how* recursive is C_1?

I do not have an answer to this specific question, but I shall propose a technical device for answering this sort of question for finite sets in general. There is a theory of randomness for *finite* strings which goes back to Kolmogorov[19]. Much of the recent work in this theory is due to G. Chaitin[20].

The principal tool used here is the length of a program. Consider a string x of symbols. Since the string is finite, there is a program P written, say, on a universal Turing machine Z which prints out x. In fact, one such program could consist of just *saying*, "print x". Such a program would contain an occurrence of x and would therefore be longer than x. However, if the string has regularities, then we can take advantage of this. A very short program could print out a string consisting of 10,000 1's.

Now, the theory goes, let us call a string *random* if the length of the shortest program to print it is roughly the length of the string itself. (This definition is almost independent of which particular Turing machine we were using.) If a string is very structured, i.e. an initial portion of some simple recursive set, then the string can be completely specified by simply giving a description of the infinite string in question together with a specification of how long an intial portion was x itself. Such a string would be highly *non*random.

The Kolmogorov-Chaitin theory is primarily concerned with defining randomness for strings, but since a finite set of strings can be coded as a single string, the theory can be adapted to sets of strings. Moreover, we need not be concerned only with the two extremes of randomness and utter simplicity. There would also be intermediate classes of strings which were not quite random, but were not simple either. By comparing the length of a string with the length of the shortest program, one could place it in an intermediate class. If the string coding a set of strings is nearly random, then deciding if a given string is in the set is primarily a memory problem. As the program gets shorter relative to the string itself, more computation and less memory are involved. This would, at least in principle, give us a means of deciding *how recursive* is grammaticality.

I shall not here develop the details of such a theory. Clearly that task belongs to a more technical paper. However I hope I have put forward a case that the task is both worthwhile and feasible.

FOOTNOTES

1. This paper has developed out of a talk entitled "Church's thesis-does the orthodox Church view need re-examination ?" given at the Boston Colloquium for the Philosophy of Science, November 18, 1975. However, my primary purpose *here* is not to question Church's thesis as much as to point out the limitations of its domain of applicability, even in discussions of effectiveness. Note that Church's thesis should really be referred to as the Church-Turing thesis, but I have used the terminology which is more prevalent in the literature.

2. For example, the discussion of effectiveness in Enderton's comparatively recent book ('A Mathematical Introduction to Logic', Academic Press, 1972, see esp. p.60) is quite close in spirit to definition 1. Hartley Rogers ('Theory of Recursive Functions', McGraw-Hill, 1967, which is stll a sort of standard work,) uses the expression "algorithmic function", and again definition 1 would be the favoured interpretation.

5. M. Dummett, the Justification of Deduction, *Proc. of the British Academy* LIX (1973)

4. Here is another example. Suppose it happens that in a certain zoo, tigers are the only striped animals. It will then be convenient to identify tigers by looking for stripes. However, if one plans to visit another zoo which might contain zebras, then one had better remember that identifying tigers with the striped animals was only a matter of convenience. (These remarks have of course no connection with whether tigers are a natural kind.) The distinction between a g-notion and an m-notion does have some distant resemblence to that between intensional and extensional equality. In the present context, 'intentional' would be a better term.

5. See for example, L. Kalmar's "An argument against the plausibility of Church's thesis",*Constructivity in Mathematics (Ed. A. Heyting)* North-Holland 1959, 72-80.

6. J. Shepherdson, Computation over Abstract Structures, *Logic Colloquium* '73 *(Ed. Rose and Shepherdson),* 445-513.(see esp. p. 446)

7. E. Engeler, On the Solvability of Algorithmic Problems, *Logic Colloquium* '73 *(Ed. Rose and Shepherdson)*, 231-252.

8. "On Formalised Computer Programs", *J. of Computer and System Sciences*, 4 (1970) 220-247. See also "Comparative Schematology", by Hewitt and Paterson, Project MAC report, M.I.T., 1970.

9. The question of effectiveness in such matters has a clear legal significance. For example the law can require us to drive on the right or the left as the case may be. As part of filling out a tax return it may require us to do some additions and multiplications. But it cannot require us to be lucky in love or in a lottery. While there is a general awareness that the notion of ethical or legal responsibility, as traditionally understood, involves free choice, the relevance of effectiveness is not as frequently pointed out. Since Church's thesis is often associated with mechanism, it is also worth remarking that in this particular context, effectiveness is not only not so associated, but rather, it forms a part of the notion of responsibility. You cannot hold someone responsible for something that they could not have helped, and hence, if the offence was one of *omission* then it must be that the action required was effectively performable. The notion of 'effective' will of course be one that is appropriate to the particular situation.

10. I assume that the reader is familiar with Turing machines. However if (s)he is not, M. Davis' 'Computability and Unsolvability', (McGraw Hill, 1958) is still an excellent reference.

11. In fact the reason we ordinarily need instructions to perform a task is that we are carrying it out for the first time, and we only remember the breaks where we need to pay special attention. After we have done it many times, we no longer need to divide it up this way and, quite often, do not even remember the individual pieces. Thus the way a task is divided into smaller pieces is rather arbitrary.

12. "Informal Rigour and Completeness Proofs", *Philosophy of Mathematics* (Ed. I. Lakatos) North-Holland 1967, 138-186.

13. I am not convinced at all that there is such a set of atomic actions. It seems far more likely that there is either no such set, or at the very least, no such usable set. At any rate, addition is not an atomic action for people as it tends to

be for computers. Thus an assertion that computers are faster than people merely because they do *arithmetic* faster, seems unwarranted. It may well be that other actions that are quite complicated for computers are in fact quite easy for people. I have myself seen a demonstration (by an engineering student from Poona University) of complicated calculations like the extraction of a fifth root to seventeen places, carried out with remarkable speed.

14. The initial work in this area is due to Floyd and Hoare. See V. Pratt, "Semantical Considerations in Floyd-Hoare Logic", *Proc. 17th IEEE Symposium on the Foundations of Computer Science,* 1976, for a complete bibliography.

15. H. Putnam, 'Some Issues in the Theory of Grammar,' *Mind Language and Reality: Philosphical Papers, Volume 2,* Cambridge University press, London, England (1975) pp. 85-107. The arguments on this particular issue are on pp. 102-105. He gives several arguments, but none of them seems to get around the problem that we do not have a theory of G as an independent set.

16. N. Chomsky (private communication) points out that there is no empirical evidence one way or another. Moreover, if one takes the point of view that the words 'both' and 'all' can appropriately be applied only to collections of cardinality two, greater than two, respectively then undecidable questions can be coded into questions of grammaticality.

17. The reason for having an oracle for a different set is to account for any thinking that goes on when one is trying to decide if a given string of symbols *is* in fact a grammatical sentence of English. If the oracle was for G itself, then one might expect all answers to be instantaneous.

18. I am not assuming that Peano Arithmetic is finitely axiomatisable. At least one of the rules will allow substitution of one formula for another to get the infinitely many induction axioms.

19. A.N. Kolmogorov, 'On the logical foundations of information theory and probability theory', *Problems of information transmission* 5,3(July-Sep. 1969) 1-4.

20. G.J. Chaitin, 'A theory of program size formally identical to information theory', *J. Assoc. Comp. Machinery* 22(1975) 329-340.