Revised May 1984

EFFECTIVE COMPUTABILITY IN ECONOMIC DECISIONS

by

R. Preston McAfee*

ABSTRACT

The theory of effective computability provides a useful handle on a wide class of economic problems. An overview of this branch of mathematical logic is provided, and applied to a variety of problems. In particular, it is shown that Rational Expectations equilibria exist under almost any sensible circumstances, in contrast to previous results.

*Assistant Professor of Economics, University of Western Ontario, London, Canada, N6A 5C2. The author wishes to thank the attendees of the Convergence to Rational Expectations Workshop, 1981, and Jeremy Greenwood, Peter Howitt, Glenn MacDonald, John McMillan and Michael Parkin for thoughtful discussions. Any errors are my own.

Introduction

This manuscript has three objectives. The first is to demonstrate that a branch of mathematical logic known as the theory of Effective Computability[1] is necessary to model some economic problems, The second is that this approach provides some quite useful and interesting results to a class of significant economic problems. Finally, I propose to introduce the reader to the simpler theorems in this area.

Toward the first objective, I will argue that the models of existence of rational expectations (RE) equilibria, as considered by Radner [6] and others, are basically inappropriate, in the sense that the equilibria found do not correspond to equilibria achievable by agents constrained to actually make the forecasts required of them. That is to say, the constraint that agents be able to actually forecast the outcome of an RE equilibria in these models cannot be satisfied. In addition, I will introduce a variant of the Prisoner's Dilemma that cannot even be modelled at all outside the context of effective computability. I think this model has a peculiar charm, but it is contrived.

Toward the second objective, I will show that, under a rather compelling assumption on the method of an economy arrives at a price outcome, rational expectations equilibria always exist. In particular, it will not be necessary to presume the existence of an auctioneer as in most models: the method taken is entirely compatible with a decentralized economy. In addition, no conditions on the dimension of the price of signal space are relevant. Further, I will solve my contrived Prisoner's Dilemma. In addition, I will make some observations on the effects of including the cost of computation into a variety of economic problems, including research and economic organization. Unfortunately the results in this case are basically unpleasant, although this is informative.

In order to achieve the second objective, I find it economical to provide an overview of the theory of effective computability, as few readers will be acquainted with it. I have limited myself to the results that have direct applications, and their antecedents as comprehensibility dictates. Nonetheless, this requires a substantial investment by the reader. I find it difficult to explain the RE theorem without reference to this development, and this yields the following format. In the next section, a variant of the Prisoner's Dilemma is proposed and discussed. In the third section, an overview of effectively computability is provided. I have labored to assume only an understanding of sets, functions and so forth, along with an understanding of countable and uncountable cardinalities. For the latter, the uninformed reader is referred to Goffman [3]. For an excellent introduction to effective computability, see Machtey and Young [5]. For more advanced methods, see Rogers [7]. The fourth section presents, discusses and solves an RE problem that is basically Radner's with the constraint that all agents in the model must be able to actually compute, in the practical sense (although unrestricted by functional form,[2] of course), their forecasts. It is interesting that restricting the space of forecast functions makes the problem more likely to have an RE equilibrium. This is not counterintuitive, as restricting the space of forecast functions also restricts what agents must compute about each other.

The fifth section suggests some other economic problems clarified by the theory of effective computability, including the optimal amount of research and the optimal organization of resources or markets. In addition, other avenues of enquiry which would likely benefit from this approach are provided, along with a list of conclusions.

Finally, I wish to add a disclaimer. The solutions to the economic problems presented in this paper would be transparent to any graduate student in mathematical logic. Indeed, the problems were posed in such a way to make the solutions consistent with an overview of effectively computability, subject to the problems retaining their economic content. The purpose of this manuscript is as much to illustrate the application of effectively computability in economic theory as to solve the problems, although the RE problem is of major economic significance.

Surprise Attack

This problem is a variant of the prisoner's dilemma, in its "one shot" form. Two neighboring nations are considering whether to attack or not. Should one attack and the other not, the former obtains a tactical advantage that leads it to conquer the latter. Otherwise, a stalemate obtains. The payoff structure is:

		Nation 2	
		Attack	Don't Attack
Nation 1	Attack	(-\$100,-\$100)	(\$100, Death)
	Don't Attack	(Death, \$100)	(\$0, \$0)

The first argument of the payoff pair is paid to Nation 1. This is the usual prisoner's dilemma, with mutual attack as the ordinary Nash equilibrium. However, these two nations are unsatisfied with perpetual war and the following alteration to the problem is proposed. Both nations will write computer programs P_i, $i=1,2$ which provide their defense strategy. These will be given to an international body which will then let P_1 "examine" P_2, that is, input P_2 to P_1, and vice versa. Afterwards, P_i provides the decision for nation i. To state this formally, P_1 will, after taking P_2 as an input, dictate the action of nation 1. Nation 2 is symmetric. A more informal way of putting this is to wonder whether the nations can choose a decision strategy which, if the decision strategies are common knowledge, allows the no attack outcome to be achieved. This coincides with asking whether no attack is a Nash equilibrium in decision strategy space.

I contend that this problem has no resolution outside the consideration of the properties of programs that can actually be written.[3] Notice that the program that dictates that one does whatever the other nation proposes to do is no strategy at all if both nations choose it. Thus, one has to design a more complicated strategy.[4] Assuming death is arbitrarily undesirable, the following definition captures what an optimal program should satisfy.

Definition: P is an optimal program for surprise attack if

i) for all programs Q, P on input Q never results in death

and ii) there does not exist an P′ satisfying (i) so that

$\{Q/P' \text{ results in mutual attack}\} \subsetneqq \{Q/P \text{ results in mutual attack}\}$.

That is, P is optimal if it avoids death at all costs, and no other program successfully avoiding death results in no attack on more inputs. Thus, we have defined P to be optimal if it pareto dominates all other programs. The following proposition resolves the issue.

Proposition: There exists a program which never results in death and, if the two nations are the same, results in no attack. There does not exist an optimal program.

This proposition solves surprise attack in the following sense. The exchange of strategies results in the existence of a strategy strictly dominating the ordinary Nash equilibrium. However, there is no best strategy, in that for any proposed strategy P, there is another that does no worse on any inputs, and strictly better on some.

As a practical aside, one could motivate surprise attack as follows. Suppose the U.S. and the U.S.S.R. agree to exchange their respective defense computer programs that direct the firing of their ICBMs. What should these programs look like? Of paramount importance in such a program, there should be no opportunity for the other nation to launch a surprise attack. In addition, one should like to avoid mutual annihilation insofar as is consistent with the first objective. That is, one should like to take advantage of the altered nature of this game to improve on the mutual attack outcome of the ordinary Nash equilibrium. The proposition resolves this insofar as is possible. The nations can improve on the Nash strategy, but there is no best strategy. The proof is actually trivial, and relegated to the appendix.

Although this problem is contrived, it does possess a certain economic content. An Nash equilibrium in an economy where decision strategies matter would normally consist of a solution where each agent acts given the strategy of the other agents. Recursive functions provide a handle on this type of problem. This, in some sense, prefigures our analysis of rational expectations, in that rational expectations requires each agent to find expectations, given the manner in which other agents compute expectations. That is, we are looking for forecast procedures which take as inputs the other agents' forecast procedure. Surprise attack is merely the rational expectations problem without any signals at all other than the forecast procedure. There is a trivial rational expectations equilibrium--always attack. However, we've shown that when the individuals are permitted to exchange their strategies, they may improve on this outcome. Consequently, they will have an incentive to do so.

Algorithms

At this point, we shall digress to consider the theory of algorithms apart from any particular economic criteria. The exposition is keyed to the eventual return to economic applications, and for this reason most of the results are presented in forms appropriate to the applications. We need to prove one theorem for insight, the others will be stated without proof.

The question of what precisely constitutes an algorithm is no longer subject to much debate. Intuitively, an algorithm is a finite set of directions which tells an agent precisely what to do under all circumstances which may arise. Just what constitutes a set of instructions depends on the language in which they are expressed, and the meaning assigned to the various terms in the language. This is, of course, not unique to the notion of algorithm. For when any intuitive idea is formally expressed, it is then an issue whether the subjective notion has been captured by the definition. Clearly one cannot prove a definition encompasses the notion, rather one can only provide evidence in the sense that one shows the definition acts like the notion.

Before continuing with the definition of an algorithm, it is of some use to consider a property of algorithms, their input/output behavior. Consider computing n factorial, $n!$. We have an input, the number n. One may compute $n!$ utilizing two memory banks, say x and k. Initially set both memories equal to n. Then use the following procedure: If $k \neq 0$, multiply x by k and then decrease k by one, and if $k = 0$, print x. This algorithm is said to compute $n!$. Observe there is a distinction between the function computed, $n!$, and the algorithm that computes it. In general we have, corresponding to every algorithm, a function which is the input/output behavior the algorithm computes. These are naturally called the computable functions, the functions that correspond to the input/output behavior of algorithms. We shall use the word program synonymously with algorithm.

Many characterizations of algorithms exist, and they are all equivalent in the sense that they compute an isomorphic set of functions. In this way, it appears that the various definitions capture the essence of algorithms. No one has been able to propose a pattern of behavior that looks like an algorithm and cannot be expressed by any of the various definitions. That every algorithmic behavior can be expressed by any of the definitions is called the Church-Turing Thesis.

It is not useful to us to actually write down a formal definition. For this, and for evidence of the Church-Turing Thesis, the reader is referred to Machtey and Young [5]. What we do need is a procedure for listing the algorithms in an effectively computable way. That is, we desire to number the algorithms with natural numbers (the set $N = \{0,1,2,\dots\}$). To do so, we need to consider an alphabet with which to express algorithms. We shall demonstrate that algorithms defined over a complex language are essentially equivalent to algorithms defined by numbers, called Gödel numbers, and which take as inputs and outputs only a single integer. This surprising result, that the complexity of languages with countably many symbols (characters) in their alphabet buys no generality, is shown by writing down an algorithmic isomorphism between the complex language algorithms and the simple ones. Thus there is an algorithm which permits us to write algorithms in one language and map these into the other language. What this demonstrates is that any function that can be computed in one language can be computed in the other. This permits us to consider the simpler case without loss of generality. The demonstration is relegated to the Appendix.

Our characterization of algorithms, to summarize, shows that an arbitrary formulation is equivalent to algorithms $A_o, A_1, \ldots$ over N. As we mentioned, these algorithms compute functions. Of course, an algorithm is not guaranteed to terminate, and in this case we adopt the convention that the functional output is ∞. Thus, for each A_n, we have a computable function φ_n: $N \rightarrow N \cup \{\infty\}$ so that the output of A_n on input $x \in N$ is $\varphi_n(x)$ if A_n terminates on this input, and ∞ otherwise. A_n on input x is written as $A_n(x)$. The input/output behavior of A_n, denoted by φ_n, are the computable (or recursive) functions.

Although our construction allows us to write φ_n as having one input, it is sometimes convenient to write φ_n as having two inputs, rather than going through the laborious decoding. This convention will be adopted freely, and the reader is reminded that we can always encode the pair into a single integer.

An example of this is the universal function, $\varphi_u(n,x)$. $\varphi_u(n,x) = \varphi_n(x)$ for all n and x. To see that φ_u exists, observe that given input n, we can reconstruct algorithmically the program corresponding to n. We may then run this program on input x, and obtain $\varphi_n(x)$. This is clearly an algorithmic procedure, so there must be a universal algorithm, and hence universal function. Consequently, we need only one algorithm to compute all algorithms.

Another example of what can be done with algorithms is called an s-m-n construction. Suppose we have a function $\varphi_k(n,x)$. Then there must exist a computable function g so that $\varphi_k(n,x) = \varphi_{g(n)}(x)$, and furthermore g is total, that is, $g(n) \neq \infty$ for all n. The reason is that given k and n, we may write the program out which, on input x, computes $\varphi_k(n,x)$. This program has a number, which is g(n). g is the function of n which encodes the program that is the kth algorithm initially provided with n.

This is a program (hence $g(n) \neq \infty$) and provides a construction of g's algorithm, showing g is computable. This demonstrates that the distinction between a program and the inputs to a program is a blurred one.

One of the fundamental results is the Recursion Theorem, also called the fixed point theorem of computable functions. It will be used to establish the general existence of rational expectations.

> Recursion Theorem: Let f be a total computable function (that is, $f(n) \neq \infty$ for all n). Then there exists an n_o so that $\varphi_{f(n_o)} = \varphi_{n_o}$.

Obviously the programs with numbers $f(n_o)$ and n_o will differ in general. Nevertheless, the Recursion Theorem shows there will be two algorithms with the property that $\varphi_{f(n_o)}$ (the output of f given the input is the program n_o) has the same input/output behavior as φ_{n_o}.

Consider writing the program described in footnote 4. Let f(n) be the program, so that, on input x, it does the following:

$$\text{Print} \begin{cases} \text{DON'T ATTACK} & \text{if } x = n \\ \text{ATTACK} & \text{if } x \neq n \end{cases}$$

It is trivial to write such a program. The Recursion Theorem, hence, guarantees there exists an n_o so that $\varphi_{n_o} = \varphi_{f(n_o)}$, that is, the program doesn't attack if and only if it meets itself.

Now consider f(n) to be the algorithm which computes A_n first, and if this terminates, prints the output plus one. Thus the Recursion Theorem implies $\varphi_{n_o} = \varphi_{f(n_o)}$. If A_{n_o} terminates, we have by construction that $\varphi_{n_o} = \varphi_{f(n_o)} = \varphi_{n_o} + 1$. Thus it is clear A_{n_o} cannot terminate, that $\varphi_{n_o} = \infty = \varphi_{f(n_o)}$. In economic situations, it would be useful to rule out the use of algorithms that do not terminate, for these cannot be cost effective computational procedures. The following theorem demonstrates that this is impossible in general. We include the proof as it will provide some intuition.

Theorem (Gödel Undecidability of the Halting Problem): It is algorithmically undecidable if an arbitrary program terminates.

Proof: By contradiction. Suppose we can write an algorithm H(n,x) that decides if $A_n(x)$ terminates. If so, we can assume

$$H(n,x) = \begin{cases} 1 \text{ if } A_n(x) \text{ terminates} \\ 0 \text{ if } A_n(s) \text{ doesn't terminate} \end{cases}$$

. Then we can write a new program of one input as follows: "compute H(n,n). If a one is obtained, do not terminate. If a zero is obtained, print zero, and terminate." It is clear this is a program, because we can write a program which doesn't terminate. Thus it has a number, say j. Now consider if H(j,j) = 1. Then by the definition of H, $A_j(j)$ terminates. By the construction of A_j, A_j in this case iterates forever. So suppose H(j,j) = 0. Then by the definition of H, $A_j(j)$ iterates forever. By the construction of A_j, A_j prints zero and terminates. Either way, a contradiction obtains. Thus no such H exists. □

The question of whether an arbitrary program terminates is called the halting problem, and we have demonstrated there is no algorithm to decide the halting problem. As a result, the halting problem is thus said to be (algorithmically or Gödel) undecidable. This is perhaps the most accessible undecidable problem known. From this, most of the wide class of undecidable problems can be constructed. Before stating these, we shall consider an intuitive reason why Gödel undecidability occurs.

Some readers may recognize Russell's Paradox embedded in the undecidability of the halting problem. Russell's paradox is based on the construction of a set of sets which are not members of themselves, and in this way is defined over all sets, including itself. Similarly, the program H is self-referential, it has to decide if H terminates, worse, programs based on H must still be determined by H. In a sense, the problem is that H falls within its own scope of analysis.

Many of the things that one might wish to compute are in fact undecidable, as the following shows.

> Rice's Theorem: Suppose $\emptyset \neq R \subsetneq N$, where R is a set of computable <u>functions</u>. Then it is undecidable if an arbitrary $n \in R$.

Rice's Theorem can be verbally expressed as follows. Suppose P is a property of computable functions, and P is nontrivial in the sense that neither all functions nor no functions display this property. Let R be the set of functions (as given by their numbers) displaying P. Then it's in general undecidable if an arbitrary algorithm computes a function with property P.

For example, undecidability of halting problem follows from P being the property that the function is never infinite. Similarly, if P is the property that the function is constant, it is undecidable if an arbitrary algorithm computes a constant (e.g. always attack).

One of the primary concerns of economists is the cost of various activities, and this applies to computation also. While specific computing devices have various properties, some powerful results follow from quite innocuous assumptions about a cost structure. Let $\Phi_n(x)$ be defined to be the cost of running A_n on input x. Assume:

i) $\Phi_n(x) = \infty$ if and only if $A_n(x)$ doesn't terminate

ii) it is decidable if $\Phi_n(x) \le \$y$.

The first assumption says that cost is infinite if and only if infinitely many resources are used. To see how we can decide if $\Phi_n(x) \le \$y$, simply run A_n on input x for \$y. If A_n has terminated, the answer is yes, otherwise no. For any reasonable model of computation, these hypotheses seem eminently plausible. Φ is called a complexity measure.

These hypotheses lead to a theorem of some economic import:

Blum Speedup Theorem: Let $g(x,y)$ be a total computable function so that $g(x,y) \le g(x,y+1)$. Then there is a total computable function f so that if $\varphi_m = f$, there is a k so that $\varphi_k = f$ and for all but finitely many x, $g(x, \Phi_k(x)) \le \Phi_m(x)$.

For example, let $g(x,y) = (xy)^2$. Then there exists a function f so that for any means of computing f, A_m, there is another way of computing f so that, for all but finitely many x, $\Phi_k(x) \le \frac{1}{x}\sqrt{\Phi_m(x)}$. For some Φ, in particular when "table look up" is inexpensive, it is possible to make the result true for all x, by patching A_k with a table of the values for the finite set.

The Blum Speedup Theorem dashes all hope of being able to generally minimize the cost of computation. Of course for some particular functions it is possible. But no matter how the cost of computation is imposed, one cannot hope for a general cost effectiveness.

There are many more results in the Theory of Algorithms that have some significance to economists. However, these results convey some of the profound implications and general flavour of the field. As we shall see in the following section, many of these results have implications in rational expectations and other areas in economic theory.

Rational Expectations Equilibrium

We will adopt the model of Radner [6], with the following constraint: any forecast procedure used by agents in the model must be effectively computable. Prior to introducing the model, it is sensible to defend this additional constraint. The motivation is as follows. If one requires agents in a model to forecast with functions that are not algorithmic, then one is requiring the agents to do the impossible. Indeed, a forecast that cannot be computed is useless--the agent cannot figure out the forecast, much less act on the forecast. That is to say, the study of RE equilibria is the enquiry into the existence of self-fulfilling expectation procedures, and when these are not procedures, they are without economic content at all.

For most economic problems, e.g. supply and demand, the imposition of computability amounts to the 'integer problem', in that one expects supply and demand to be computable. Indeed, if the profit function is computable, profit maximization yields the computation of the supply function, and so on, and the smooth real analysis model will approximate the discrete computable function model arbitrarily close, with virtually no loss at a great gain in elegance. However, when computability constrains expectation functions, the character of the results alters entirely. That is, when all agents are using functions to analyze all other agents' functions, the self-referential nature of this problem makes computability significant.

This section will demonstrate that, under arbitrary conditions, RE equilibria exist. The definition coincides with that of Radner, with the restriction of all functions to be computable. Second, some of these equilibria have a peculiar characteristic in that all agents may spend an infinite (unbounded) amount of time computing the equilibria. Finally, no procedure exists for separating the models with this peculiar (and undesirable) attribute from those that don't.

There are I agents, indexed by $i=1,\dots,I$, and they wish to forecast a distribution of a random variable $p \in P$. Each individual receives a drawing or a private information value $x \in X$. Both P and X are countable sets. Given that agent i forecasts the joint distribution of p,x_i to be $\varphi_{n_i}(p,x_i)$, the true distribution of (p,x_i) will be $\sigma_i(n_1,\dots,n_I,p,x_i)$ where n_i identifies φ_{n_i}. σ_i may depend on the agent in question because the meaning of $x \in X$ could be different for distinct agents. The $(n_1,\dots,n_I)$ is a RE equilibria if

(1) $\qquad \varphi_{n_i}(p,x_i) = \sigma_i(n_1,\dots,n_I,p,x_i) \qquad\qquad i=1,\dots,I$

We now assume σ_i is algorithmic, that is, an algorithm computes σ_i. If this assumption fails, then (1) cannot hold, because φ_{n_i} is assumed algorithmic. The plausibility for this assumption is simply seen. Embedded in σ is the notion of market clearing, and how forecast prices affect the true outcome. Usually, forecast prices are used in maximizing expected profits, which result in firm outputs. The sum of these outputs is entered into the inverse demand functions to obtain the true price, with a stochastic element perhaps. This is, up to the stochastic element, algorithmic. The profit maximization is explicitly solved by the firms in some fashion and is thereby algorithmic. Demand is algorithmic as well in that there must be some procedure for computing it; otherwise it would not be possible to decide how much to buy. The σ may represent a relatively centralized function, as in the excess demand function, $z(p,s) = 0$, used by Radner [6], or may represent a quite decentralized procedure whereby markets clear partially at a local level, and then differences in outcomes are exploited by arbitrage. In either case, procedures exist for finding the value of p up to a stochastic element, forcing σ to be algorithmic.

Theorem 1: If σ_i is algorithmic, there exists $n_1,\ldots,n_I$ so that (1) obtains.

Proof: By the s-m-n theorem (Rogers [7], theorem V, p. 23), we may equivalently write (1) as (suppressing x,p):

$$(2) \qquad \varphi_{n_i} = \varphi_{f_i(n_1,\ldots,n_I)} \qquad\qquad i=1,\ldots,I.$$

By the Recursion Theorem (Rogers [7], theorem III, p. 181) we obtain a function $n_1^*(n_2,\ldots,n_I)$ so that

$$(3) \qquad \varphi_{n_1^*} = \varphi_{f_1(n_1^*,n_2,\ldots,n_I)}.$$

Write:

$$\varphi_{f_i(n_1^*,\ldots,n_I)} \equiv \varphi_{f_i^2(n_2,\ldots,n_I)}.$$

Using the Recursion Theorem, we obtain $n_2^*(n_3,\ldots,n_I)$ and

$$\varphi_{n_2^*} = \varphi_{f_2^2(n_2^*,n_3,\ldots,n_I)}.$$

Inductively, then, we let $f_i^k(n_k,\ldots,n_I) = f_i(n_1^*,n_2^*,\ldots,n_{k-1}^*,n_k,\ldots,n_I)$ and obtain

$$(4) \qquad \varphi_{n_k^*} = \varphi_{f_k^k(n_k^*,n_{k+1},\ldots,n_I)} \qquad\qquad k=1,\ldots,I.$$

Then, if $\hat{n}_I = n_I^*$ and $\hat{n}_k = n_k^*(\hat{n}_{k+1},\ldots,\hat{n}_I)$, equations (3) and (4) are equivalent to:

$$\varphi_{\hat{n}_i} = \varphi_{f(\hat{n}_1,\ldots,\hat{n}_I)}, \qquad i=1,\ldots,I.$$

This is precisely (2), as desired.

□

This theorem is quite general, but unfortunately cannot distinguish the two cases of importance. Consider the simple example where p is a monetary aggregate, $X = \emptyset$, and the policy is to let agent 1 forecast p, and then make the forecast wrong. This is analogous to the time consistency problem considered in Kydland and Prescott [4].
Thus, σ is constructed so that, as soon as agent 1 makes a forecast, he is guaranteed to be wrong. Clearly agent 1 will never make a forecast in this case, and that way will be correct. Because the value of p cannot come out until agent 1 makes a decision, if agent 1 indefinitely delays, then p will never come out. Thus agent 1 is correct in his "empty" forecast.

There is another way of viewing this as well. Consider a game with two agents, one of whom plays second. Rational expectations on the part of the first guarantees that it is not possible for the second player to exploit the beliefs of the first player about the second player. In many games, however, the ability to play second always allows one to capitalize on the first player's beliefs. The money supply game is an example. The first player does have a rational expectations counter strategy to the second player's exploitation of the first player's beliefs, which is to delay. If the second player follows the rule of 'exploit the first player's belief', he must also wait until the first player acts.

There is an interesting interpretation of this, relevant to the bond market. If the monetary authority follows a rule that invariably wipes out the return obtained by buyers of long-term bonds, the RE policy for potential bond buyers is to never decide to purchase bonds. Consequently, the long-term bond market never opens. This provides a somewhat different view of time consistency than existed before. In most RE models, it was assumed that RE implied that individuals could act as if they knew the distribution of future inflation rates. Consequently, the time consistency problem arose, as once these individuals commit themselves by purchasing a bond, the government's optimal policy may require fooling them. The solution is that, if the government plays second, they delay indefinitely.

This case is to be contrasted to the case when individuals know the rule, and decide not to buy. In that case, the individuals know that the government will fool them, and so choose not to buy. Consequently, they have reached a decision, and so the government may then play. This is different from keeping the government in suspense, by not reaching a decision. There is no difference in purchasing, but there is a behavioral difference on the part of the government, whose role requires that they wait until the individuals make a decision. If the individuals never predict an outcome, and no outcome occurs, then the individuals trivially display RE. If the individuals don't predict an outcome, but decide not to act, the government may then play, and make the individual's lack of prediction incorrect.

An element of self-reference separates these two cases. If the individual tries to analyze the government's policy, which functionally depends on the individual's policy, the individual is forced to analyze himself, and hence analyze his analysis of himself, and so forth. In the other case, he knows the government will fool him, and so decides not to buy bonds. Thus, the individual does not bother to analyze himself. The latter may be optimal, but it is not rational expectations. As the purpose of this paper is merely to demonstrate the possibility of RE, the fact that the individual can defeat the policy that the second player always fools the first is sufficient.

Evidently we would like to distinguish the cases when forecasts actually are made, which can be considered as the market opening, and the case where no forecasts, and hence no actual outcome of p, occur. The following theorem demonstrates this is not possible in general.

Theorem 2 (Kleene): There is no effective procedure for ascertaining whether arbitrarily chosen σ_i will result in a forecast.

Proof: This is Theorem VIII, p. 26 in Rogers.

□

This theorem shows that there is no effective decision procedure for splitting the σ's which open the market from those that do not. One particular class of σ will open the market, however. Consider the σ's that must terminate in a fixed amount of time. That is, if the agents fail to make forecasts in T units of time, a distribution $\sigma(\phi,\ldots,\phi,x,p)$ will emerge anyway. Then, clearly, the RE equilibrium will not involve infinite delay.

To make this point clear, consider a simple model wherein agents forecast a price, p_f^i, produce $y_i(p_f^i)$ that maximizes profit, and then actual price is $p(\sum_{i=1}^{I} y_i(p_f^i))$. RE occurs if $p_f^j = p(\sum_{i=1}^{I} y_i(p_f^i))$, as there are no random disturbances. Now, suppose that if an agent does not produce by time T, y_i is zero. Then, if the agent fails to forecast, a price still emerges. Thus, not making a forecast is not RE, so RE will involve a forecast.

It should be mentioned that the procedures used by agents in Theorem 1 will not, in general, be optimal. The purpose of Theorem 1 was to demonstrate that rational expectations is possible, not that RE is optimal. Indeed, the only circumstances where RE appears to be optimal occurs when the cost of computation is zero. An advantage of taking the procedural component of expectation formation into account is the ability to explicitly include the cost of computation in the optimization. This advantage is weakened by the extreme complexity of computational cost measures. This is discussed further in the next section.

This section demonstrates that the hypothesis that expectation formation procedures be computable implies the existence of RE equilibria in general. Unfortunately, unless bounds are placed on the length of time computation may occupy, in some cases RE may require eternal computation. In addition, it is not possible in general to distinguish models where the agents plot forever, which is analogous to the market not opening, from those that do.

Conclusion

In this section, we shall present some further considerations in applying effective computability to economic decisions. These are presented at a more intuitive and speculative level, in comparison to the analyses of rational expectations and surprise attack.

Consider the notion of rationality. Let us say, for example, that a firm is rational if it maximizes profit. From Rice's Theorem, we know that it is undecidable if an arbitrary algorithm maximizes profit. There will thus be algorithms where it is not possible to establish if they are rational. Rationality is hence undecidable in general. Note that any property can be substituted for rationality, as long as it is possible that a function displays this property, and possible that it doesn't.

As another application, recall the Blum Speedup Theorem, which says that for some functions, it is not possible to compute in an optimal way.

Thus our hope of profit maximizing subject to the cost of computation (among other costs) is not in general realizable. One might hope to add the cost of obtaining the minimum cost computation to remove this difficulty, but of course there will be a Blum Speedup Theorem for this adjusted cost structure.

This should not appear too surprising. In general we should expect that, to ascertain the optimal amount of computation one should do involves doing too much, and even then one doesn't know if one stopped just short of a great breakthrough. This makes the optimal amount of research undecidable in general. That there should always be an incredibly cheaper way of doing the same thing is perhaps a bit more startling.

In this paper, we have not written down a definition of algorithmic behavior, which has simplified the theoretical presentation. The following application depends on a model of algorithms and consequently shall not be as precisely stated as might be desired. We shall provide the intuition only.

Algorithms work by having rules of manipulation which are applied in a systematic fashion to a state of a system. Economic structures operate in a similar fashion. There is an initial state, and such operations as production, consumption, transportation and exchange occur. That these correspond to the operation of algorithms may be observed. Production corresponds to calculation, where inputs are operated on to achieve an output. Consumption might correspond to erasing a file, while transportation and exchange change attributes (location and ownership) of files. Thus, an economy in action is not unlike a program being operated; indeed, most empirical work depends on this analogy.

This mapping leads to several observations. There must be economic structures for which we cannot decide if they have the same feasible set (To see this, consider the set: $\{n / \varphi_n = \varphi_k\}$ for some fixed k. It's undecidable if an arbitrary i is in the set.) In fact, the results we have obtained apply not only to the processing of information, but to general economic operations. Suppose we consider a set of economic procedures corresponding to production of goods, and write "programs" with these procedures. If the initial set of procedures is sufficiently complex (just how complex is beyond the scope of this paper, but it is not unreasonably complex), we can obtain the set of algorithms through it. Imposing a cost structure on these procedures such as is imposed for the Blum Speedup Theorem yields: In general, there will be achievable outcomes for which there is no cheapest way of reaching that outcome for all but finitely many initial states. That is, in general, cost minimization will not always be possible in real economic processes.

This is intuitively appealing. We are aware that it is not feasible to turn electricity into physical motion with 100% efficiency, as some energy is transformed into heat. However, there is no reason to believe that there is a most efficient motor either. Essentially we have argued that there is not necessarily a most efficient economy for a similar reason, for under some circumstances there is no cheapest way of doing a given transformation. This may be viewed as following from the inherent discreteness algorithms entail. That such a phenomenon is provably a feature of a large class of economic structures (essentially those with a minimal complexity) is perhaps interesting.

To try to sum up, imposing the restriction that agents actually be able to compute a forecast procedure may narrow the agents' choices significantly. This is particularly true of models of the world where how an agent thinks about the world has an effect on how the world comes out. Despite this, we can guarantee the existence of rational expectations, although it will in general not be decidable if a particular decision rule is in fact a rational expectations rule. This fact rules out some things both modelers, and agents in the model, might desire to do.

One further related topic concerns the computational complexity of decision procedures. While it may be possible in principle to solve a given problem, the cost may be prohibitive. Many intractable problems have been identified, and it is likely they appear in Economic Theory as well. One such problem is found in the theory of auditing Bailey et al [2].

FOOTNOTES

[1]Recursive Function Theory, the Theory of Algorithms and the Theory of Effective Computability are synonymous.

[2]Anderson and Sonnenschein [1] provide an RE example where agents can actually forecast the prices, by using least squares, which we can clearly compute. Unlike that paper, this paper considers all possible computable forecast procedures, as it is possible for an RE forecast procedure to fail to be a least squares method.

[3]One obviously runs into an infinite regress in the usual approach to this problem. Observe also that this model is similar to the choice of a strategy in a supergame, except that one does not exactly choose the strategy given the other's strategy. Rather, one chooses a metastrategy that maps the other person's strategy choice into an outcome.

[4]One might desire to use a strategy which attacks unless the other person is using the same strategy, in which case it (and hence both) don't attack. To do this, one has to show that one can write a strategy that can recognize itself. We shall prove that one can, and it is precisely this fact that is difficult to even contemplate outside the domain of effective computability. One might have hypothesized that a program which recognizes itself must, of necessity, be longer than itself, as it would appear to have a copy of itself embedded in itself. This reasoning is incorrect.

REFERENCES

[1] Anderson, R., and Sonnenschein, H. "Rational Expectations Equilibrium with Linear Models," Econometric Research Program Research Memorandum No. 286, 1981.

[2] Bailey, A., McAfee, R. P., and Whinston, A. "An Application of Complexity Theory to the Analysis of Internal Control", Auditing: A Journal of Theory and Practice," 1, no. 1, pp. 38-52.

[3] Goffman, C. Real Functions, Prindle Weber and Schmidt, Inc.: New York, 1953.

[4] Kydland, F.,and Prescott, E., "Rules Rather than Discretion: The Inconsistency of Optimal Plans," <u>Journal of Political Economy</u> 85, no. 3, 1977.

[5] Machley, M., and Young, D. <u>An Introduction to the General Theory</u> of Algorithms, North-Holland, New York, 1978.

[6] Radner, R. "Rational Expectations Equilibrium: Generic Existence and the Information Revealed by Prices," <u>Econometrica</u> 47, no. 3, 1979.

[7] Rogers, M. <u>The Theory of Recursive Functions and Effective Computability</u>, McGraw Hill, New York, 1967.

APPENDIX

Demonstration that General Algorithms are Isomorphic to Gödel Numbered Algorithms over N

The actual construction of this mapping may appear to be tedious detail, but it serves three purposes. First, it provides some insight into a major method of the theory of algorithms, which is to actually display an algorithm that performs a desired job, to prove that job can be done. Second, it will provide some contact with algorithmic processes, which reminds us how radically different computable functions are from real analysis functions. Finally, a startling result is proved, namely that it is entirely irrelevant to the computability issue whether a large number of inputs exist or a single one does. When one considers how much more complex an economic universe is when a second good is added, one sees the power inherent in this construction.

Consider a possibly infinite alphabet $a_1, a_2, a_3, \ldots$. First, we wish to represent words in this alphabet with numbers (the reason will become apparent). We shall let a string of n "1"s mean a_n. To write $a_n a_m$, write n "1"s followed by a zero and then followed by m "1"s. In general, to write the word $a_{n_1} a_{n_2}, \ldots, a_{n_k}$, write n_1 "1"s, then a zero, then n_2 1's, then a zero, and so forth. To separate words, put two consecutive zeros. In this way, an algorithm written in the language comprised of an infinite alphabet $a_1, a_2, \ldots$ can be expressed as a sequence of zeros and ones. That is to say, we can consider our algorithm to be written in a binary code, because in that code we may express the more complex alphabet $a_1, a_2, \ldots$.

Some of the numbers in the binary code will not be meaningful, that is, they will correspond to gibberish when expressed in the original language. The simplest way of dealing with this is to say that if an expression (in our language or in the binary code) does not correspond to a meaningful set of instructions, it is said to correspond to an algorithm which does nothing. To see why we can say this, consider running programs on a computer. If a program is written which is not meaningful in the computer language, the computer just halts, perhaps with an error message. In the same vein, we shall let things which are not algorithms merely not be processed, and either print "error" or any other term of our choice. These may be useless or trivial algorithms, ones with no meaningful directions, but they may be considered to be algorithms nonetheless. As a result, these non-programs become programs (or algorithms) in our revised view, by establishing a "default option".

Thus, we can number the algorithms with natural numbers: $A_0, A_1, A_2, \ldots$. The number of the n^{th} program is the binary representation of the directions and is called a Gödel number. Clearly the inputs can also be transformed into numbers by the same procedure, so that we need to only consider algorithms which take tuples of integers as inputs. Of course, we can also make the output into tuples of integers, again by the binary mapping. We will consider A_n and the representation of the code, n, to be synonymous.

We have spent some time detailing a method of obtaining Gödel numbers for algorithms because it is of importance that this procedure be algorithmic. The description has revealed an algorithm to produce the Gödel number from a program, and vice versa, and thus we know the procedure is algorithmic. Consequently, we can write an algorithm to translate integer inputs and the Gödel number into a character string input and an algorithm, respectively, in the original language. Then we may do the computation, and translate the output back into numbers using the binary coding procedure. For this reason, the set of algorithms over an arbitrary language with countably many characters is equivalent to a programming system where programs are integers and inputs and outputs are strings of integers.

Actually, this is weaker than need be. Consider the "pairing function", $< , >$, given by:

$$<m,n> = 2^m(2n+1) - 1, \quad m,n \in N$$

You may verify $< , >$ maps $N \times N$ into N bijectively. As a result, a pair of integers may be encoded into a single integer. To encode tuples of integers of arbitrary length, consider:

$$E(x_1,\ldots,x_n) = <n, <x_1, <x_2, <\ldots<x_{n-1}, x_n>\ldots>$$

Given a single number, we may decode a tuple by first decoding n, which is the number of times two divides $E(x_1,\ldots,x_n)+1$ evenly. From the remainder, $\frac{E(x_1,\ldots,x_n)+1}{2^n}$, we may find x_1, using a similar procedure. We do this n times, which we can do because n was the first number observed. The remainder at this point is $2x_n + 1$, from which we decode x_n. Consequently, we may at will translate tuples of integers of arbitrary (finite) length into a single integer. Thus, we may consider our algorithms to take a single integer as an input and produce a single integer as an output, because that corresponds to the world where tuples are taken as inputs and outputs by an algorithmic mapping. This greatly simplifies notation, without loss of generality, for we know that algorithms which take a single integer input, and are described only by a number, and produce a single integer output can be algorithmically transformed into the seemingly more complex world where character strings describe the algorithm, inputs and outputs. The reader is reminded that the theorems we shall state have equivalent formulations in the more complex language.

Before continuing, it is useful at this time to consider the cardinality of sets. This will illustrate an important difference between algorithmic functions and general real valued functions. A finite set is said to have the cardinality of the number of members it has. That is, a set A has cardinality n if and only if there is a function $f: A \rightarrow \{1,\ldots,n\}$ bijectively. In the same way, a set B is said to be countably infinite (or of the cardinality of the natural numbers) if there is a bijection $f: B \rightarrow N$. (A set is said to be countable if it is finite or countably infinite.) It is a fact, proved by Cantor, that the real numbers have a cardinality larger than N (Goffman [1953]).

Now consider the pairing function $<,>$ considered earlier, $<,>$ is a bijection from NxN into N. The set of pairs of natural numbers comprise the union over the first component of copies of N. This proves the countable union of countable sets is countable. Because the number of algorithms is countable (the Gödel number yields the mapping) and the inputs form a countable set (using our integer mapping), the set of outputs is a countable union (over algorithms) of countable sets (over inputs) and is hence countable. Thus not all real numbers can be computed, in fact, the set of reals which can be computed form a set of measure zero in the ordinary reals. This will be meaningful when we consider the place of algorithms in the usual economic theory. Thus, no real functions can be everywhere computed, and hence the usual forecast procedures presumed in economic models are not computable.

<u>Proof of the Proposition</u>:

By the existence of a universal function, there is a k so that

$$\varphi_k(n,j) = \varphi_j(n).$$

By the s-m-n theorem, there is a total function g so that

$$\varphi_{g(n)}(j) = \varphi_k(n,j).$$

Thus, there is a total function h(n) so that

$$\varphi_{h(n)}(j) = \begin{cases} \varphi_{g(n)}(j) & n \neq j \\ \text{"No Attack"} & n = j \end{cases}$$

By the Recursion Theorem, there is an n* so that

$$\varphi_{n^*} = \varphi_{h(n^*)}$$

Thus, for $j \neq n^*$,

$$\varphi_{n^*}(j) = \varphi_{h(n^*)}(j) = \varphi_{g(n^*)}(j) = \varphi_k(n^*, j) = \varphi_j(n^*)$$

and for $j = n^*$,

$$\varphi_{n^*}(j) = \varphi_j(n^*) = \text{"No Attack"}.$$

This proves the first line of the proposition. The second follows immediately from the above construction, by patching in a "No Attack" for input n* to another program as calculated above.