Non-Computable Strategies and Discounted Repeated Games[*]

John Nachbar
Department of Economics
Washington University in St. Louis

and

William R. Zame
Department of Economics
UCLA

April 1995

WORKING PAPER NUMBER 735

[*]This work was begun while Nachbar was a visitor at The Center for Mathematical Studies in Economics and Management Science at Northwestern University; he is grateful for their hospitality. We are also grateful to Robert Anderson and Neil Gretsky and to seminar audiences at UCLA for useful comments, and to the National Science Foundation and the UCLA Academic Senate Committee on Research for financial support. This paper is an outgrowth of work reported in "Learning and Computability in Discounted Supergames."

Abstract

A number of authors have used formal models of computation to capture the idea of "bounded rationality" in repeated games. Most of this literature has used computability by a finite automaton as the standard. A conceptual difficulty with this standard is that the decision problem is not "closed." That is, for every strategy implementable by an automaton, there is some best response implementable by an automaton, but there may not exist any algorithm for *finding* such a best response that can be implemented by an automaton. However, such algorithms can be always be implemented by a Turing machine, the most powerful formal model of computation. In this paper, we investigate whether the decision problem can be closed by adopting Turing machines as the standard of computability. The answer we offer is negative. Indeed, for a large class of discounted repeated games (including the repeated prisoner's dilemma) there exist strategies implementable by a Turing machine for which *no* best response is implementable by a Turing machine.

JEL Classification C72, C73

Keywords repeated games, bounded rationality, computable strategies, Turing machines

1 Introduction

Following a suggestion in Aumann (1981), a number of authors have used the notion of computability to model "bounded rationality" in repeated games. In most of this literature, "computability" is taken to mean computability by a finite automaton (more accurately, Moore machine); Rubinstein (1986) represents seminal work in this genre (we discuss the literature more fully below). In this paper, we investigate some consequences of adopting a more sophisticated standard, computability by a Turing machine (an idealized computer with a finite description but infinite memory). For infinitely repeated games with discounting, we show that even the adoption of the Turing machine standard (arguably the most sophisticated computation standard possible) leads to fundamental "incompleteness" results: in general, there will exist computable strategies admitting no computable best responses, and the problem of choosing a best response (even when computable best responses do exist) does not have a computable solution. We believe these results have serious implications for the analysis of repeated games.

Our focus on computability by a Turing machine as a standard is not at all arbitrary. It is generally accepted in the mathematical and computational literature (although see below) that the notion of partial recursive function is the correct embodiment of the informal notion of computable function, a point of view usually referred to as Church's Thesis.[1] Interest in Turing machines stems in part from the fact that every partial recursive function can be computed by a Turing machine, and every function computable by a Turing machine is partial recursive; thus, Turing machines can compute "everything that can be computed" - at least if we accept Church's thesis.[2] By contrast, finite automata can compute only a proper subset of the partial recursive functions. Binmore (1987), for example, argues on this basis that Turing machines provide the "right" computational model of the absolute

[1]The partial recursive functions are those which can be built up from the identity function, the projection function, the zero function and the successor function by composition, combination, exponentiation and repetition; see Odifreddi (1987), for example.

[2]Turing machines are also attractive because of their close resemblance to modern digital computers. Neural networks represent quite a different computational model, and are attractive in part because they resemble human brains, but they have precisely the same computational power as Turing machines.

limits of rational play.

What does it mean to insist on computability in game theory? One possibility is that we insist that strategies themselves be computable, but make no requirements about how players are to arrive at these strategies. (This limits the complexity of strategies but not the rationality of players.) Another possibility is to insist that the decision problems of players actually be computable. (This limits the rationality of players.) We establish "incompleteness" results for each of these interpretations. Specifically, we show first that for a large class of two-player stage games (including the familiar prisoner's dilemma) there exist computable (i.e., Turing machine implementable) strategies which do not admit computable best responses.[3,4] Although the strategies we use are necessarily somewhat complicated, they meet the minimal test of being part of an equilibrium — indeed, of a subgame perfect equilibrium. That is, there exist pure behavioral strategies s_1, s_2 with the following properties:

- the pair (s_1, s_2) constitutes a subgame perfect equilibrium
- s_2 is computable[5]
- *no* best response to s_2 is computable

As a reader familiar with computability will anticipate, our construction involves building a version of the "halting problem" into the strategy of Player 2. The construction is slightly delicate because, with discounting, it is *almost* true that a player can determine what to play after any history simply by optimizing over a long enough finite horizon.

Second, we show for the same large class of stage games that players' decision problems do not have computable solutions. That is, there is no Turing machine which, given the description of a Turing machine used by

[3]The particular structure we assume for the stage game simplifies our arguments substantially, but we conjecture that similar results will hold for nearly all infinitely repeated games.

[4]Of course, in the discounted framework, best responses always exist.

[5]In fact, s_2 can be computed by a Turing machine that requires only polynomial time to perform its calculations.

one player, computes a best response for the other player. Of course, this follows from our previous result, since a computable solution to the decision problem would necessarily produce a computable strategy. In fact, we show that there is no computable solution to the decision problem for a player *even if we insist that the opponent choose among computable strategies which* do *admit computable best responses.*

We point out that our results apply only to *exact* best responses, and not to *approximate* best responses. Indeed, given a computable strategy s_2 and any $\varepsilon > 0$, it is easy to find a computable strategy s_1 which is an ε-best response. (That is, no response to s_2 improves on s_1 by more than ε.) Moreover, the procedure for finding such a computable ε-best response is computable (simply optimize over a sufficiently long finite horizon). Thus, the computability problems we identify do not necessarily have important consequences for *payoffs*. However, these computability problems do have important consequences for *paths of play*, because the path of play specified by an ε-equilibrium may be very different from that specified by an exact equilibrium. In a sense, strategies implementable by Turing machines are intrinsically vulnerable to certain types of "trembles" which can cause observed play to depart substantially from what one would predict if players truly optimized.

Of course, our results do not bear on the *existence* of subgame perfect equilibria in computable strategies. (Indeed, the Folk Theorem can be derived using only computable strategies; see, for example, Ben-Porath and Peleg (1987).) However, our results have some relevance for attempts to formalize how (boundedly) rational players might come to play a Nash (or subgame perfect) equilibrium in the first place. Computability problems will become still more severe if players have non-degenerate beliefs, as would typically be the case in learning or evolutionary environments; see Nachbar (1995).[6]

Given the negative character of our conclusions, it is natural to enquire whether we have chosen the wrong model of computation. Perhaps Turing machine computability is either too restrictive or too permissive. We consider

[6]In the present paper, beliefs are degenerate, in the sense that each player places probability one on a single strategy of its opponent.

these in turn.

Turing implementability is too weak. As we have noted earlier, the Turing computable functions are exactly those which are partial recursive, the mathematical benchmark for "computable." There are other models of finite computation, but all are either equivalent to Turing machines, in the sense of also implementing exactly the partial recursive functions, or weaker, in the sense of implementing a proper subset of the partial recursive functions. What about models of *non-finite* computation? At this writing, there is disagreement as to whether non-finite computation is possible even in principle. Penrose (1989) represents a leading attempt to argue for the possibility of non-finite computation, but while we are sympathetic to Penrose's point of view, it is sobering to realize that his argument rests ultimately on conjectures about an as-yet-unformulated quantum theory of gravity.

Turing implementability is too strong. If we weaken the computability standard to computability by a finite automaton, then the problem of implementing a best response disappears: for every strategy implementable by a finite automaton, there is a best response which is implementable by a finite automaton (indeed, by a finite automaton of the same "complexity"); see Stanford (1989). However, the decision problem still has no computable solution: no finite automaton can compute a best response (or even an ε-best response, for $\varepsilon > 0$ small) to every possible finite automaton. Solving the decision problem requires a Turing machine (or at least some machine with infinite memory).[7]

It is sometimes argued that computability by a finite automaton is more "realistic" than computability by a Turing machine, but we question this argument in the context of infinitely repeated games. The only difference between a Turing machine and a finite automaton is that a Turing machine has unbounded memory; for both machines the set of *instructions* is required to be finite. In an infinitely repeated game, it seems strange to us to rule out the possibility of adding additional memory. It should also be kept in mind that, in an infinitely repeated game, unbounded memory is already

[7]Indeed, one might have hoped that enlarging the computability standard from finite automaton to Turing implementability would "close" the model; our results show that this is not the case.

required in order to keep track of the history. Of course, one might insist on limits to the computational complexity required in each period, but the Turing machines we require are quite efficient: the number of steps (hence the amount of memory) required by one of our machines in order to compute the action in the n-th period grows only as a polynomial in n. We conjecture that our results could be obtained using only a Turing machine for which the number of steps required to compute the action in the n-th period grows only linearly in n. Such a machine would require, in the n-th period, memory of only the same order as that needed to record the history up to that point. It seems very hard indeed to justify restricting attention to machines simpler than this.

Our results are closely related to those of Knoblauch (1994), who considers computability of best responses in the context of the infinitely repeated prisoner's dilemma with payoffs evaluated as long run averages, rather than discounted present values as here. When payoffs are calculated as long run averages, there are computable strategies which do not admit any best responses at all — computable or not. Knoblauch shows something stronger: there are computable strategies which *do* have a best response, but for which no best response is computable. Our paper provides the parallel result for a large class of discounted repeated games.

More generally, our paper contributes to research on computer implementation of repeated game strategies. There is a fairly large literature on automaton implementability in repeated games; in addition to papers already cited, see Kalai and Stanford (1988). The papers in the automaton literature closest in spirit to the present exercise are Gilboa (1988) and Ben-Porath (1990), who investigate the complexity of the procedure for choosing a best response in undiscounted games with automaton implementable strategies. In particular, Gilboa shows that the procedure for computing a best response can be implemented by a polynomial-time Turing machine provided, as here, that the game and opposing automaton are known in advance.

By contrast, relatively little work has been done on Turing implementability in repeated games. Aside from Knoblauch (1994), some other papers in the general area include Megiddo and Wigderson (1987), Gilboa and Samet (1989) (a hybrid, combining both automaton and Turing implementability), Fornow and Whang (1993), and Anderlini and Sabourian (1993).

Finally, our paper complements a line of research, begun by Binmore (1987) and pursued by Anderlini (1990) and Canning (1992), in which the idea of a player "thinking through" a game is explicitly formalized by modeling each player's decision procedure as a Turing machine which receives as input a description of the game and the program encoding the other player's decision procedure.[8] Because of the built-in self-reference (each player decides what to do based on his calculation of what the other player decides to do, based on ...), computability problems arise even for finite games. In the present paper, we assume away this particular self-reference; in our formulation, a player has beliefs directly over its opponent's *strategies*, rather than over its opponent's *decision procedures*. In our formulation, no computability problems would arise for finite games.

Following this Introduction, Section 2 provides the basic facts about recursive functions, Turing machines and computability. Although we refer to the literature for details, the material provided in Section 2 should provide enough information for the neophyte. Section 3 recalls the basic facts about repeated games with discounting. Section 4 presents a computable strategy which admits no computable best response. Section 5 establishes that there can be no algorithm for computing best responses, even when computable best responses exist.

[8]This basic setup had been employed earlier by McAfee (1984) for a different purpose.

2 Turing Machines

A Turing machine is an idealized computation device with a finite description but unbounded memory. We offer here a brief description of Turing machines and their operation. The treatment we give is informal, and not intended to be complete, but we hope it will provide adequate background, even for the uninitiated. As will become clear, precise knowledge of the structure and operation of Turing machines is not necessary for understanding the remainder of the paper. In part this is because we follow a convention, standard in the computation literature, of asserting that a Turing machine can be constructed whenever it is "clear" that the relevant calculations can be carried out as a finite sequence of discrete steps. This elision, which saves an immense amount of detail, is based on an appeal to the fact that the set of Turing computable functions is exactly the same as the set of flowchart computable functions; see Theorem 1.7.12 in Odifreddi (1987). Readers who want a more comprehensive and systematic treatment of Turing machines are directed to Hopcroft and Ullman (1979) or Cutland (1982).

A Turing machine consists of

(i) an infinite processing tape which is divided into squares

(ii) a read/write head

(iii) a processing unit with a finite number of internal states

Each square on the tape may be marked with either 0, 1 or b (blank).[9]

The action of a Turing machine may be described in the following way. At the beginning of a computation, the read/write head is positioned over some square of the tape, say x_0, and the tape is filled from x_0 rightwards with a finite string (possibly empty) of 0's and 1's (i.e., a binary string), the *input*. The remainder of the tape is filled with b's. At the beginning of step k in its computation, the machine is in some state q and the read/write head

[9]Although it is usual to work with a three-symbol alphabet, an alphabet of two symbols would in fact suffice; of course modern digital computers use a two-symbol alphabet. However, allowing for blank squares avoids some unpleasant technicalities.

is positioned over some square x. According to the current state and the content of the square, four things are determined:

(i) whether the current square is to be overwritten (with a 0, 1 or b) or left alone

(ii) what the machine's new internal state will be

(iii) whether the read/write head is to remain where it is, or to move one square along the tape to the left, or to move one square along the tape to the right

(iv) whether the machine is to continue or to halt (stop)

We identify the completion of these four items as a single computational *step*. Note that we allow for the possibility that the machine does not halt. If the machine does halt, the *output* of the computation is the binary string (possible empty) beginning at the read/write head and extending leftward until the first b. Since each computational step alters at most one square, and the number of computational steps executed before halting is finite, the output of any computation is finite.

We write $(0+1)^*$ for the set of all finite binary strings of 0's and 1's. The input of a Turing machine is a string $w \in (0+1)^*$; presented with such an input, the machine M either halts with some output written on the tape or fails to halt. If we write W for the set of inputs on which M halts, then the Turing machine M *computes* (or *implements*) a function

$$\varphi_M : W \rightarrow (0+1)^*$$

If M halts on all inputs then $W = (0+1)^*$ and we say that the function φ_M is *total*; otherwise, φ_M is *partial*. Let Φ_{TM} be the set of functions computed by Turing machines. Since each Turing machine has a finite description, Φ_{TM} is countably infinite. Of course, the set of all (partial) functions on $(0+1)^*$ is uncountably infinite, so "most" functions cannot be computed by a Turing machine.

There are standard schemes for using binary strings to encode (i.e., unambiguously represent) natural numbers, paired natural numbers (hence rational numbers), formal logical expressions, English words, and so forth. Of course, the *same* binary string will have different interpretations for different codings. The statement in the introduction that Turing machines can compute any partial recursive function means in particular that, with the proper coding, we can construct a Turing machine able to execute standard arithmetical operations such as addition, subtraction, multiplication, and raising to a power.

We will be interested in whether certain sets can be enumerated by a Turing machine. We say that a set $W \subset (0+1)^*$ is ***recursively enumerable*** if there is a Turing machine M which maps the natural numbers $\mathbb{N} = \{0, 1, \ldots\}$ (properly encoded, of course) onto W. We say that a set $W \subset (0+1)^*$ is ***recursive*** if both W and $(0+1)^* \setminus W$ are recursively enumerable. Typically, we will be interested in W interpreted as a subset of $\mathbb{N}$; in this case, W is recursive if and only if both W and $\mathbb{N} \setminus W$ are recursively enumerable.[10]

Because a Turing machine has a finite description, it is possible to encode this description itself in binary; write m for the binary code of the machine M. If the coding is done properly, it will be "machine readable" in the following sense: There exists a Turing machine U such that $\varphi_U(m, \cdot) = \varphi_M(\cdot)$ (so that the two functions have the same domain and have equal values on that domain) for every Turing machine M.[11] Any such machine U is called a *universal* Turing machine; infinitely many universal Turing machines exist. The action of U on (m, w) is not different in principle from the familiar action of a general purpose computer running a software implementation of a special purpose computer (such as a dedicated word processor). The novelty here is that, because memory (the tape) is unbounded, U can simulate *any* Turing machine no matter how complicated, including U itself. By contrast, a computer with bounded memory (an automaton) can simulate another computer only if the other computer is sufficiently small, in an appropriate sense.

It is not hard to show that the set of Turing machine codes is recursively enumerable (indeed recursive). Because of this, one can identify the set of Turing machine codes with $\mathbb{N}$ in a computable manner. In particular, one

[10]This actually requires proof (because of coding technicalities).

[11]Of course, (m, w) refers to a binary string which encodes the pair of strings m and w; as usual, we skip over exactly how this coding is done.

can show that there exists a universal Turing machine U such that $\varphi_U(n, \cdot) = \varphi_{T_n}(\cdot)$, where T_n is the Turing machine identified with integer n.[12]

The proof of our main theorem hinges on the nature of the following sets:

$$\begin{aligned} A &= \{n \in \mathbb{N} : T_n \text{ halts on input } n \text{ and } \varphi_n(n) = 0\} \\ B &= \{n \in \mathbb{N} : T_n \text{ halts on input } n \text{ and } \varphi_n(n) \neq 0\} \end{aligned}$$

For our purposes the crucial facts about the sets A, B are that they are recursively enumerable but cannot be separated by a computable function (and in particular are not recursive). It is convenient to separate the (quite standard) arguments into two lemmas.

Lemma 1 *The sets A and B are recursively enumerable.*

Proof We sketch the argument for A and omit the parallel argument for B.

Note first that A is infinite. To see this, we construct an infinite family of Turing machines which always halt and write 0. For each integer p consider the machine which has p internal states, and operates in the following way. The machine begins operation in state 1. Given any input, the machine neither moves its read/write head nor writes, but transits to state 2; in state 2, it neither moves its read/write head nor writes, but transits to state 3, and so forth. In state p the machine writes 0, leaving the read/write head over the initial square x_0, and halts. Clearly the index of this machine belongs to the set A, and these machines are all distinct (because they have different numbers of states), so A is infinite.[13]

We now fix a universal Turing machine U. We construct a machine S which operates according to the following algorithm. On input p, the machine S "zigzags" through the list of Turing machines by using U to compute the

[12]That the Turing machines can be put in 1-1 correspondence with $\mathbb{N}$ follows simply from the fact that the set of Turing machines is countably infinite. The strength of the claim above is that this correspondence can be implemented by a Turing machine. Since there are countably infinite subsets of $(0 + 1)^*$ (and of $\mathbb{N}$) which are not recursively enumerable, this assertion is not trivial.

[13]As we see here, a Turing machine is *not* characterized by the (partial) function it computes. Indeed, each of this infinite family of Turing machines computes the 0 function.

first step of machine T_0 on input 0, then the first two steps of T_0 on input 0, then the first step of T_1 on input 1, then the first three steps of T_0 on input 0, then the first two steps of T_1 on input 1, then the first step of T_2 on input 2, and so on. S halts the $(p+1)$-st time that it finds a machine, say n_p, that halts with an output of 0; S then returns the output n_p.

To see that S enumerates A, we must show that φ_S is a total function and that its image is precisely A. Consider therefore the operation of S on the input p. Since A is infinite, we can choose $p+1$ indices $n_1, \ldots, n_{p+1} \in A$. By definition, the Turing machine T_{n_k} halts on input n_k; let $r(n_k)$ be the number of steps executed before halting. When given the input p, the machine S will eventually simulate $r(n_1)$ steps of the machine T_{n_1} and $r(n_2)$ steps of the machine T_{n_2}, ..., and $r(n_{p+1})$ steps the machine $T_{n_{p+1}}$, and will halt at that point, if it had not already halted before. Hence S is total. Clearly every output of a computation by S is an element of A. Finally, the output of S will be n_k as soon as S simulates $r(n_k)$ steps of the operation of T_{n_k}, so S enumerates all the elements of A. □

Lemma 2 *There is no computable* total *function* $\varphi : \mathbb{N} \to \{0,1\}$ *such that*

$$\varphi(n) = \begin{cases} 1 & \text{if } n \in A \\ 0 & \text{if } n \in B \end{cases}$$

Consequently, neither A nor B nor $A \cup B$ is recursive.

Proof Suppose such a total computable function φ existed. Let T be a Turing machine that computes φ and let n_T be the natural number encoding T. Since φ is total, the machine T halts on the input n_T, so its index n_T belongs to A or to B. If $n_T \in B$ then $\varphi(n_T) = 0$, but by our coding $\varphi_{n_T}(n_T) = \varphi(n_T) = 0$, whence $n_T \in A$, which is a contradiction. On the other hand, if $n_T \in A$ then $\varphi_{n_T}(n_T) = \varphi(n_T) = 1$, whence $n_T \in B$, which is a contradiction. We conclude that no such total computable function φ can exist.

To see that A is not recursive, suppose that it were; let φ be a computable function which enumerates A and let φ' be a computable function which enumerates $\mathbb{N} \setminus A$. Define a computable function ψ as follows: In order to compute the value of ψ at n, compute the sequence of numbers $\varphi(1), \varphi'(1), \varphi(2), \varphi'(2), \ldots$ to the point where n first appears. Define $\psi(n) = 0$ if $n = \varphi(k)$ for some k, and $\psi(n) = 1$ if $n = \varphi'(k)$. (This defines a computable function precisely because φ and φ' are both computable and total, and enumerate A and its complement $\mathbb{N} \setminus A$.) Since A and B are disjoint, the function φ separates them. Since we have shown that no such separating function exists, we conclude that A is not recursive.

A similar argument shows that B is not recursive. To see that $A \cup B$ is not recursive, note that an enumeration of $\mathbb{N} \setminus A \cup B$ and an enumeration of B could be interleaved to provide an enumeration of $\mathbb{N} \setminus A$, contradicting the fact that A is not recursive. □

The requirement in Lemma 2 that φ be *total* is crucial; it is easy to use the recursive enumerability of A and B to find computable *partial* functions which separate A and B. We note that $A \cup B$ is the set of indices of Turing machines that halt on their own index; the failure of this set to be recursive is a formal expression of the fact that the halting problem for Turing machines (that is, the determination of which Turing machines halt) is unsolvable.

A Turing machine M is usually considered to be (relatively) simple if there is a polynomial function $g : \mathbb{N} \to \mathbb{N}$ such that, for every natural number k, the machine M executes at most $g(k)$ steps for any input string of length k on which M halts. Turing machines which satisfy this condition are said to be *polynomial-time bounded.*

3 Repeated Games and Computable Strategies

We consider infinitely repeated 2-player games with discounting. Let $G = (A_1, A_2, u_1, u_2)$ be the *stage game*, so that A_i is the finite *action set* and $u_i : A_1 \times A_2 \to \mathbb{N}$ is the *payoff function* of player i.[14] A finite *history* of length T is an element of the T-fold cartesian product

$$\mathcal{H}^T = (A_1 \times A_2) \times \ldots (A_1 \times A_2)$$

(There is a unique empty history of length zero.) We write $\mathcal{H} = \bigcup_{T \geq 0} \mathcal{H}^T$ for the set of all finite histories. For $h \in \mathcal{H}^T$ and $0 < t \leq T$, the action profile that occurred at date t is denoted $h[t]$ and is the projection of h onto its t coordinate.

A *path of play* is an infinite sequence of action profiles. The set of paths of play is denoted $\mathcal{Z}$. Given a path of play $z \in \mathcal{Z}$, the projection onto the t coordinate is denoted $z[t]$, and the projection onto the first T coordinates is denoted $z(T)$. Note that $z(T) \in \mathcal{H}^T$ is a history of length T.

We write G^∞ for the game which consists of infinitely many repetitions of the stage game G. In each period, players are informed of the actions taken in earlier periods, so a *pure strategy* for player i in the infinitely repeated game G^∞ is a function $s_i : \mathcal{H} \to A_i$ which assigns an action following each finite history. We write S_i for the set of pure strategies for player i and $S = S_1 \times S_2$ for the set of *pure strategy profiles*. A pure strategy profile $s \in S$ determines a path of play $z_s \in \mathcal{Z}$. If the discount factor is δ then the payoff to player i given players follow the strategy profile (s_1, s_2) is:

$$V_i(s_1, s_2) = \sum_{t=1}^{\infty} \delta^{t-1} u_i\left(z_s[t]\right)$$

As usual, we say that the pure strategy s_1 is a *best response* to the pure strategy s_2 if

$$V_1(s_1, s_2) \geq V_1(s_1', s_2)$$

[14]The restriction to payoffs which are natural numbers is designed to forestall trivial computability problems.

for every pure strategy s_1'. Since the set of pure strategies is compact in the product topology the payoff function V_1 is continuous in the product topology (because future payoffs are discounted), best responses always exist. We write $\mathrm{B}_1(s_2)$ for the set of best responses for player 1 to pure strategy s_2.

The set of finite histories can be identified with $\mathbb{N}$ in a computable manner (the construction is similar to that used to encode Turing machine descriptions). We may also identify the action set A_i of player i with a subset of $\mathbb{N}$. We may therefore view a pure strategy for player i as a (total) function on $\mathbb{N}$ such that for any history h (encoded as a natural number), $s_i(h)$ is the natural number encoding the action taken at h. We say that a pure strategy *can be implemented by a Turing machine* if the function $s_i : \mathbb{N} \rightarrow \mathbb{N}$ can be computed by a Turing machine.

In view of our earlier remarks about the capacity of Turing machines to compute anything that is computable, we shall identify a "computable" strategy as one that can be implemented by a Turing machine. Assuming that player i has at least two actions, the set of all pure strategies for player i is uncountable. On the other hand, the set of strategies implementable by a Turing machine is countable. In particular, there are many strategies that are *not* implementable by a Turing machine.

4 Non-Computable Strategies

Throughout this section we restrict attention to stage games that satisfy the following assumption. (This assumption greatly facilitates our argument, but is probably not necessary for our conclusions; see Nachbar (1994).)

Assumption A There are action profiles $(x, X), (y, Y) \in A_1 \times A_2$ such that

- (y, Y) is a Nash equilibrium of the stage game G
- the payoff from (x, X) strictly Pareto dominates the payoff from (y, Y)

Prisoner's Dilemma is a familiar example of a game satisfying this assumption.

Our principal result is:

Theorem 1 *Assume the stage game G satisfies Assumption A. Then there is a rational number $\underline{\delta} > 0$ such that for every rational discount factor δ with $\underline{\delta} < \delta < 1$, there is a (pure strategy) subgame perfect equilibrium profile (s_1, s_2) of the infinitely repeated game G^∞ for which:*

(a) s_2 can be implemented by a Turing machine

(b) no best response to s_2 can be implemented by a Turing machine

Before proceeding with the details of our construction, it may be helpful to outline briefly the basic logic. The strategy s_2 and the set of best responses to s_2 will be (almost) completely described in terms of (1) a *prescribed path*, the path the players are supposed to follow, (2) a relatively mild punishment path P^1, and (3) a harsher punishment path P^2. (Our description is not quite complete because of a complication stemming from the fact that certain types of deviations are not immediately punished. We shall be careful about this complication in the proof proper, but we simply ignore it for the moment.)

The prescribed path involves an infinite number of "test dates," each followed by a "reward period" lasting K^r dates, followed in turn by an "adjustment period" lasting K^a dates. The first date is a test date, so tests occur at dates $1, 2 + K^r + K^a, \ldots, 1 + n(1 + K^r + K^a), \ldots$ The strategy s_2 is constructed so that no Turing machine for player 1 can compute correct play at *every* test date. If player 1 chooses incorrectly at some test date, then play will eventually switch to the punishment path P^1. (This part of our construction is similar to that of Knoblauch (1994).) However, because the strategy s_2 is required to be implementable by a Turing machine, computability difficulties (for player 2) entail that deviations at test dates by player 1 may not be detected immediately. Hence, the transition to the punishment path will not in general be immediate; if player 1 chooses incorrectly at date t, the transition to P^1 will occur at date $t + \tau$, and the length of the delay τ will depend on t.

This unavoidable delay in the transition to the punishment path entails two difficulties:

(i) Because future payoffs are discounted, the present value of a delayed punishment will be small.

(ii) Because punishment is not immediate, a player whose deviation at a test date has not yet been detected has an incentive to deviate again.

The first of these difficulties is addressed by the adjustment periods. During adjustment periods, player 2 gives player 1 a high payoff if player 1's payoff to date has been low and a low payoff if player 1's payoff to date has been high. Because of this adjustment, player 1's maximum discounted gain from deviating at a test date is outweighed by the future punishment, no matter how long the punishment delay. The second of these difficulties is addressed by our use of reward periods and two punishment paths. A deviation at a test date — which might remain undetected for a long period of time — is punished by reversion to the relatively mild punishment P^1, but a deviation at any other date — which will be detected immediately — is punished by reversion to the harsh punishment P^2. If player 1 has deviated at the test date t, he will still prefer to follow the prescribed play until the punishment date $t + \tau$ and suffer reversion to the punishment path P^1, rather than to deviate at any intermediate date and suffer an immediate reversion to P^2.

We now proceed to fill in the details.

Proof of Theorem 1 As sketched above, s_2 and $\mathrm{B}_1(s_2)$ will be characterized in terms of a prescribed path of play and two punishment equilibria, P^1 and P^2.

The prescribed path will consist of test periods at dates $1, 1 + (1 + K^r + K^a), 1 + 2(1 + K^r + K^a), \ldots, 1 + n(1 + K^r + K^a), \ldots$, interspersed by $K^r \in \mathbb{N}$ reward periods and $K^a \in \mathbb{N}$ adjustment periods. We proceed recursively. Let $\mathcal{H}^t_{\text{nodev}} \subset \mathcal{H}^t$ be the subset of t-period histories such that neither player has yet deviated from the prescribed path. $\mathcal{H}^t_{\text{nodev}}$ will typically not be a singleton, since s_2 will allow Player 1 to play either of two different actions at an infinite subset of the test periods. We will also have to consider histories in which player 1 may have deviated in one or more test periods, but for which punishment will not begin for at least another period. Formally, let $\mathcal{H}^t_{\text{test}} \subset \mathcal{H}^t$ be the subset of t-period histories such that the only deviations that have occurred (if any) are deviations by player 1 in test periods, and none of these deviations will be punished earlier than date $t + 2$. Of course, $\mathcal{H}^t_{\text{nodev}} \subset \mathcal{H}^t_{\text{test}}$.

Let $A, B \subset \mathbb{N}$ be the sets defined in Section 2. If date $t + 1 = 1 + n(1 + K^r + K^a)$ is a test period and the current history $h^t \in \mathcal{H}^t_{\text{nodev}}$ then prescribed play is according to the following rules (where X, x, y are the actions specified in Assumption A):

1. (x, X) if $n \in A$;

2. (y, X) if $n \in B$;

3. either (x, X) or (y, X) if $n \notin A \cup B$.

(The alternative, that $h^t \in \mathcal{H}^t_{\text{test}} \setminus \mathcal{H}^t_{\text{nodev}}$, presents a bit of problem, to which we will return below.) If player 1 fails to play either x or y in a test period, or if player 2 fails to play X, then play switches next period to P^2. P^2 will simply be reversion to Nash equilibrium: (y, Y) forever.

Because A and B are recursively enumerable (Lemma 1), if player 1 deviates during a test period, player 2 will be able to detect the deviation in finite time by stepping through the enumeration of A and B; we will specify below exactly how player 2 does this. However, by Lemma 2, there is no Turing implementable function φ on the natural numbers such that $\varphi(n) = 1$ if $n \in A$ and $\varphi(n) = 0$ if $n \in B$. It will then follow that player 1's best response cannot be Turing implementable, since if it were, the implementing Turing machine could be used to construct a Turing machine which implemented φ. We will record this observation formally at the conclusion of the overall proof.

The Turing machine implementing s_2 checks for test period deviations during test and adjustment periods, but not during reward periods. The method for checking is as follows. In the beginning of period $2 + K^r$, which is the first adjustment period, player 2's Turing machine simulates the first instruction of Turing machine 0 operating on input 0. If, as a result of this simulation, Player 2's Turing machine finds that Turing machine 0 halts on input 0, then player 2 compares player 1's action in the first period with $\varphi_0(0)$. If $\varphi_0(0) = 0$ and player 1 played x, then player 2 judges that player 1 played correctly and so player 2 plays according to the prescribed path in the current period $2 + K^r$. Similarly if $\varphi_0(0) \neq 0$ and player 1 played y. If player 1 is found to have deviated, on the other hand (say, if $\varphi_0(0) = 0$ but player 1 played y), then both players switch to P^1 in the current period, period $2 + K^r$.

In period $3+K^r$, assuming no deviation is detected in period $2+K^r$, player 2's Turing machine executes the first two instructions of Turing machine 0 operating on input 0 (or executes until Turing machine 0 halts, whichever comes first). And so on. At date $t = 1 + n(1 + K^r + K^a) + K^r + T$, where $T \leq K^a + 1$, if no deviation has yet been detected, player 2's Turing machine executes the first $n(1 + K^a) + T$ instructions for Turing machine 0. It *also* executes the first $(n - 1)(1 + K^a) + T$ instructions for Turing machine 1, the first $(n - 2)(1 + K^a) + T$ instructions for Turing machine 2, and so on for the first $n + 1$ Turing machines (or executes for each machine until the machines halts, whichever comes first). If $\varphi_n(n)$ is defined, then eventually the Turing machine implementing player 2's strategy detects this, but there is no *a priori* bound on when this detection occurs. If Player 2 detects multiple deviations, the punishment is still just P^1. A virtue of this test procedure (which is essentially the same as that in Knoblauch (1994)) is that the calculations performed by player 2 increase at most polynomially, even if $\varphi_n(n)$ is undefined.

We have omitted discussing what happens if $h^t \in \mathcal{H}^t_{\text{test}} \setminus \mathcal{H}^t_{\text{nodev}}$, which is a history off the prescribed path. In a standard Folk Theorem construction, such histories would be of little consequence; players would immediately switch to a punishment path (assuming they weren't on one already). However, we do not have this luxury, because for such histories the Turing machine implementing player 2's strategy won't *know* that play is off the prescribed path. We handle this as follows. Until a deviation is detected, s_2 directs player 2 to play as though no deviation has yet occurred. In particular, if $t+1$ is a test date, then s_2 directs player 2 to play X. We likewise require player 1 to continue to play as prescribed at reward and adjustment dates. However, we have little leverage over player 1 at test dates. To ensure subgame perfection, we assume that player 1's strategy directs player 1 to choose optimally at test date $t+1$, but we do not specify what this optimum is. Note that, under s_2, player 2 will respond to the first detected deviation from the rule "x if $n \in A$, y if $n \in B$." This may not be the first deviation committed by player 1.

We next turn to the adjustment periods. It is convenient to consider discounted payoffs in current terms. For $h \in \mathcal{H}^t_{\text{test}}$, let $\pi(h)$ denote the value of player 1's payoffs over the first t periods, discounted forward to date $t+1$. Thus

$$\pi(h) = \frac{1}{\delta^t} \sum_{i=1}^{t} \delta^{i-1} u_1(h[i])$$

Let $\Pi^t = \{\pi \in \mathbb{R} : \pi = \pi(h) \text{ for some } h \in \mathcal{H}^t_{\text{test}}\}$. Since Π^t is finite, it has a maximal element, $\overline{\pi}^t$, and a minimal element, $\underline{\pi}^t$.

Let $\underline{u}(a_1, a_2) = \min_i u_i(a_1, a_2)$ and $\overline{u}(a_1, a_2) = \max_i u_i(a_1, a_2)$. Let

$$\underline{u} = \min_{a_1 \in A_1, a_2 \in A_2} \underline{u}(a_1, a_2)$$

and

$$\overline{u} = \max_{a_1 \in A_1, a_2 \in A_2} \overline{u}(a_1, a_2)$$

Let $\eta = \overline{u} - \underline{u} > 0$.

If date $t+1$ is an adjustment period, prescribed play following a history $h \in \mathcal{H}^t_{\mathbf{test}}$ is

1. (x, X) if $\pi(h)^t - \underline{\pi}^t \leq \eta$.

2. (y, Y) otherwise.

If either player deviates from this prescribed play, play switches in the next period $(t+2)$ to the punishment equilibrium P^2, (y, Y) forever.

The following lemma records the fact that play in the adjustment periods shrinks the payoff differential $\overline{\pi}^t - \underline{\pi}^t$ by a discrete amount whenever the differential lies between η and 3η. Lemma 4 below will then verify that $\overline{\pi}^t - \underline{\pi}^t$ is indeed always less than 3η for δ large enough.

Lemma 3 *There is a* $\delta_1 \in (0,1)$ *and an* $\alpha > 0$ *such that for any (rational)* $\delta \in (\delta_1, 1)$*, if* $t+1$ *is an adjustment period and* $\eta < \overline{\pi}^t - \underline{\pi}^t < 3\eta$*, then*

$$(\overline{\pi}^t - \underline{\pi}^t) - (\overline{\pi}^{t+1} - \underline{\pi}^{t+1}) > \alpha$$

Proof. Choose δ_1 to be any number in the interval

$$\left(1 - \frac{u_1(x,X) - u_1(y,Y)}{3\eta}, 1\right) \subset \left(\frac{2}{3}, 1\right)$$

Since $\eta < \overline{\pi}^t - \underline{\pi}^t$ and $t+1$ is an adjustment period, $\overline{\pi}^{t+1} \leq (\overline{\pi}^t + u_1(y,Y))/\delta$. (The inequality could be strict if the history corresponding to $\overline{\pi}^t$ contains a deviation detected for the first time at the beginning of date $t+2$.) Similarly, $\underline{\pi}^{t+1} \geq (\underline{\pi}^t + u_1(x,X))/\delta$. Then

$$\begin{aligned}
(\overline{\pi}^t - \underline{\pi}^t) - (\overline{\pi}^{t+1} - \underline{\pi}^{t+1}) &\geq (\overline{\pi}^t - \underline{\pi}^t) - \frac{1}{\delta}[(\overline{\pi}^t - \underline{\pi}^t) + (u_1(x,X) - u_1(y,Y))] \\
&\geq \frac{1}{\delta}[(u_1(x,X) - u_1(y,Y)) - (\overline{\pi}^t - \underline{\pi}^t)(1-\delta)] \\
&> \frac{1}{\delta}[(u_1(x,X) - u_1(y,Y)) - 3\eta(1-\delta)] \\
&> \frac{1}{\delta}[(u_1(x,X) - u_1(y,Y)) - 3\eta(1-\delta_1)]
\end{aligned}$$

The right-hand side is strictly positive provided $\delta_1 > 1-(u_1(x,X)-u_1(y,Y))/(3\eta)$, as assumed. Then, since $\delta < 1$,

$$(\overline{\pi}^t - \underline{\pi}^t) - (\overline{\pi}^{t+1} - \underline{\pi}^{t+1}) > \alpha$$

where $\alpha = (u_1(x,X) - u_1(y,Y)) - 3\eta(1-\delta_1) > 0$. □

Fix the value of δ_1 in accordance with the previous lemma. The bound $\alpha > 0$ tells us how many adjustment periods it would take to reduce a maximum deviation of 3η to a number below η, a decrease of 2η. Specifically, set

$$K^a = \left\lceil \frac{2\eta}{\alpha} \right\rceil$$

where $\lceil x \rceil$ is the smallest integer greater than or equal to x. K^a depends on stage game payoffs, but not on δ.

Before establishing Lemma 4, which justifies the assumption above that $\overline{\pi}^t - \underline{\pi}^t < 3\eta$ for all t, we must first specify behavior during reward periods. In reward periods, players play (x,X) K^r times in succession. If either player ever deviates, then play switches immediately to P^2. K^r is set to ensure that player 1 has no incentive to deviate during any adjustment period and player 2 has no incentive to deviate during any period, test, reward, or adjustment, *even if player 1 has already deviated at a test period, so that play eventually switches to P^1*. We choose K^r as follows. Suppose that it is the beginning of date t and that an incorrect "answer" by player 1 at some earlier test date will cause a switch to P^1 at date $t+\tau$. Suppose further that a test period is included amongst the τ periods (the other possibility, that the next τ periods do not contain a test period, is considered in the proof of Lemma 6). Then, by our construction, all K^r reward periods subsequent to that test period must be included as well, since deviations are never detected during a reward period. Therefore, $\tau \geq 1 + K^r$. Take K^r so that it satisfies

$$\begin{aligned}
\underline{u} + K^r u_1(x,X) + 2K^a(y,Y) &> (1 + K^r + 2K^a)\left(\tfrac{2}{3}u_1(x,X) + \tfrac{1}{3}u_1(y,Y)\right) \\
\underline{u} + K^r u_2(x,X) + 2K^a(y,Y) &> (1 + K^r + 2K^a)\left(\tfrac{2}{3}u_2(x,X) + \tfrac{1}{3}u_2(y,Y)\right)
\end{aligned}$$

The left-hand side gives the undiscounted worst case, which occurs when t is the first adjustment period in a sequence of K^a adjustment periods, so that

τ includes a test period, a sequence of K^r reward periods, and *two* sequences of K^a adjustment periods. Then

$$K^r > 4K^a + \max\left\{\frac{2u_1(x,X) + u_1(y,Y) - 3\underline{u}}{u_1(x,X) - u_1(y,Y)}, \frac{2u_2(x,X) + u_2(y,Y) - 3\underline{u}}{u_2(x,X) - u_2(y,Y)}\right\}$$

Take any such $K^r \in \mathbb{N}$. Then it is easy (if somewhat tedious) to confirm that there is a $\delta_2 \in [\delta_1, 1)$ such that for any $\delta \in (\delta_2, 1)$, for any period t, if the current history is in $\mathcal{H}^{t-1}_{\text{test}}$, if play switches to P^1 at date $t + \tau$, and if τ includes a test period, then the average payoff to player i over the next τ periods is at least $(u_i(x,X) + u_i(y,Y))/2$. By continuity, if no deviation has in fact yet occurred, so "$\tau = \infty$," then the average payoff to player i is also at least $(u_i(x,X) + u_i(y,Y))/2$.

With K^r thus defined, we can now verify that the maximum payoff difference $\overline{\pi}^t - \underline{\pi}^t$ is indeed less than 3η for δ large enough, which was the hypothesis in Lemma 3.

Lemma 4 *There is a $\delta_3 \in [\delta_2, 1)$ such that for any (rational) $\delta \in (\delta_3, 1)$, $\overline{\pi}^t - \underline{\pi}^t < 3\eta$ for all t.*

Proof. The argument is by induction on test periods. At the beginning of period 1, the differential $\overline{\pi}^0 - \underline{\pi}^0$ equals 0. During the next K^r reward periods, play is (x, X) for all histories in $\mathcal{H}^t_{\text{test}}$. Therefore, at the start of the first adjustment period, the differential is at most

$$\frac{\eta}{\delta^{1+K^r}}$$

This is less than 3η provided $\delta_3 \geq (1/3)^{1/(1+K^r)} > 1/3$.

Note next that if period $t + 1$ is an adjustment period and $\overline{\pi}^t - \underline{\pi}^t \leq \eta$, then prescribed play is (x, X) for all histories in $\mathcal{H}^t_{\text{test}}$; hence

$$\overline{\pi}^{t+1} - \underline{\pi}^{t+1} \leq \frac{\eta}{\delta}$$

The right-hand side is less than 3η provided $\delta_3 \geq 1/3$.

Next suppose the induction hypothesis ($\overline{\pi}^t - \underline{\pi}^t < 3\eta$) holds for test n, which occurs at date $t = 1 + n(1 + K^r + K^a)$. From Lemma 3, the definition of

K^a, the induction hypothesis, and the last observation, we have $\overline{\pi}^t - \underline{\pi}^t < 3\eta$ for each $t = 1 + n(1 + K^r + K^a), \ldots, 1 + (n+1)(1 + K^r + K^a) - 1 = (n+1)(1 + K^r + K^a)$. Moreover, by the definition of K^a, at the beginning of the $(n+1)$ test period, $t = 1 + (n+1)(1 + K^r + K^a)$:

$$\overline{\pi}^{(n+1)(1+K^r+K^a)} - \underline{\pi}^{(n+1)(1+K^r+K^a)} < \frac{\eta}{\delta}$$

Hence at the beginning of the first adjustment period after test $n+1$ (i.e., at date $t = 2 + K^r + (n+1)(1 + K^r + K^a)$)

$$\overline{\pi}^{1+K^r+(n+1)(1+K^r+K^a)} - \underline{\pi}^{1+K^r+(n+1)(1+K^r+K^a)} < \left[\frac{\eta}{\delta} + \eta\right] \frac{1}{\delta^{1+K^r}}$$

The polynomial equation $3\delta^{1+K^r} - \delta - 1 = 0$ has a solution in $((1/3)^{1/(1+K^r)}, 1)$; call it δ^*. Then the right-hand side, while greater than 2η, is less than 3η provided $\delta_3 = \max\{\delta_2, \delta^*\}$. □

To complete our description, we must fill in two missing details. First, we specify that the punishment equilibrium P^1 consists of play of (y, Y) for 3 periods, followed by one period of (x, X), followed by another 3 periods of (y, Y) and so on. If either player ever deviates, player reverts to P^2 forever. The average payoff of P^1 is thus

$$\tilde{v}_i^{P^1}(\delta) = u_i(x, X)\frac{\delta^3(1-\delta)}{1-\delta^4} + u_i(y, Y)\left[1 - \frac{\delta^3(1-\delta)}{1-\delta^4}\right]$$

For δ close to 1, this is approximately $u_i(x, X)\frac{1}{4} + u_i(y, Y)\frac{3}{4}$. In contrast, the average payoff along the prescribed path of play is at least $u_i(x, X)\frac{1}{2} + u_i(y, Y)\frac{1}{2}$ while the average payoff along P^2 is $\tilde{v}_i^{P^2} = u_i(y, Y)$.

Second, and last, we must specify what happens following certain "bad" histories in which both players deviate. Following such histories, we suppose both players switch to P^2.

We now must verify that we have indeed described a subgame perfect equilibrium strategy profile (or actually set of profiles, because of the flexibility player 1 has at tests n such that $n \in (A \cup B)^c$).

By the usual Folk Theorem argument, there is a $\delta_4 \in [\delta_3, 1)$ such that for $\delta \in (\delta_4, 1)$, P^1 is a subgame perfect equilibrium. P^2 is a subgame perfect equilibrium, since it is simply repeated play of a Nash equilibrium. It remains to check that neither player has any incentive to deviate along the prescribed path.

Lemma 5 *There is a $\delta_5 \in [\delta_4, 1)$ such that for $\delta \in (\delta_5, 1)$, Player 1 has no incentive to deviate at a test date, if no deviation has yet occurred.*

Proof. Suppose that the test date is t and that punishment will begin at date $t+\tau$. Then in date $t+\tau$ terms, the maximum payoff gain from deviating is

$$3\eta + \frac{1}{1-\delta}\left[\tilde{v}_i^{P^1}(\delta) - \left(u_i(x,X)\frac{1}{2} + u_i(y,Y)\frac{1}{2}\right)\right]$$

Choose δ_5 high enough so that this expression is negative. □

As for test dates when player 1 has previously deviated, recall that at such test dates, we have stipulated that the player chooses optimally. This optimal choice will be either x or y (although which we cannot say) for the same reason, given below, that player 2 will choose X at every test date.

Lemma 6 *There is a $\underline{\delta} \in [\delta_5, 1)$ such that for $\delta \in (\underline{\delta}, 1)$, neither player has any incentive to deviate, following any history in $\mathcal{H}^t_{\text{test}}$, if deviation would result in a switch to P^2 the following period.*

Proof. Explicitly, player 1 has no incentive to deviate in any reward or adjustment period, and player 2 has no incentive to deviate in any period, test, reward, or adjustment. To see this, suppose that the date is t and that play switches to P^1 at date $t+\tau$. If τ does not include a test period, then the lowest possible *average* payoff from not deviating further is $(1 - \delta^{K^a})u_i(y,Y) + \delta^{K^a}\tilde{v}_i^{P^1}(\delta)$. Thus, the maximum possible average payoff gain from deviating is less than

$$(1-\delta)\overline{u} + \delta u_i(y,Y) - \left[(1-\delta^{K^a})u_i(y,Y) + \delta^{K^a}\tilde{v}_i^{P^1}(\delta)\right]$$

We can find $\delta_6 \in [\delta_5, 1)$ such that for $\delta \in (\delta_6, 1)$ this expression is negative.

Suppose, on the other hand, that τ does include a test period; hence $\tau \geq 1 + K^r$. Then the the lowest possible average payoff from not deviating further is $(1-\delta^{1+K^r})(u_i(x,X)+u_i(y,Y))/2 + \delta^{1+K^r}\tilde{v}_i^{P1}(\delta)$ which is strictly greater than the lowest possible payoff when τ does not include a test period. Hence δ_6 will work here as well. Finally, note that this argument applies also in the limit as τ goes to ∞, which corresponds to the remaining case in which player 1 has not deviated in a previous test period. Therefore, set $\underline{\delta} = \delta_6$. □

It follows from Lemmas 5 and 6 that we have indeed specified a set of subgame perfect equilibrium profiles. It remains only to verify that none of player 1's best responses are Turing implementable. Suppose player 1 had a best response s_1 which was implemented by a Turing machine M. For any $n \in \mathbb{N}$, $s = (s_1, s_2)$ defines a unique history $h \in \mathcal{H}_{\text{nodev}}^{n(1+K^r+K^a)}$, namely $z_s(n(1+K^r+K^a))$. Since both s_1 and s_2 are Turing implementable, one can show that there exists a Turing machine, call it G, which implements the *total* function $\varphi_G : \mathbb{N} \to \mathbb{N}$

$$\varphi_G(n) = z_s(n(1+K^r+K^a))$$

Represent x as 1 and y as 0. Then

$$\varphi_M(\varphi_G(n)) = \begin{cases} 1 & \text{if } n \in A \\ 0 & \text{if } n \in B \end{cases}$$

Then there is a Turing machine D such that $\varphi_D : \mathbb{N} \to \{0,1\}$ is a *total* function defined by $\varphi_D(n) = \varphi_M(\varphi_G(n))$. But this contradicts Lemma 2. This concludes the proof of the Theorem. □

As we have noted in the Introduction, the strategy s_2 can be implemented by a Turing machine which operates in polynomial time (that is, requires at each period t a number of steps which is bounded by a polynomial in t). Although this is reasonably self-evident for most of our construction, the number $\underline{\pi}^t$, used to define behavior in the adjustment periods, represents a potential difficulty. In fact, however, $\underline{\pi}^t$ has a simple, polynomial-time characterization. Let $\underline{a}_1 = \text{argmin}_{x,y}\{u_1(x,X), u_1(y,Y)\}$. It is easily verified by induction that $\underline{\pi}^t$ is the date $t+1$ present value of the payoff stream to player 1 generated by that element of $\mathcal{H}_{\text{test}}^t$ in which player 1 plays $\underline{a}_1$ at every test date at which the correct action is not computed by date $t+1$.

Using this characterization, one can confirm that $\underline{\pi}^t$ can be calculated in polynomial time, and hence that s_2 itself can be computed in polynomial time.

5 The Decision Problem

Theorem 1 demonstrates the existence of computable strategies that do not admit a computable best response. In this section we show that, even if we restrict our attention to computable strategies that *do* admit a computable best response, the problem of *finding* such a best response may not have a computable solution.

This decision problem may be formalized in the following way. Let C be the set of strategies for player 2 which are computable and let $C' \subset C$ be the subset of computable strategies which admit a computable best response. By definition, each strategy $s_2 \in C'$ can be implemented by a Turing machine, and admits a best response s_1 which can also be implemented by a Turing machine. An *effective procedure* for computing a best response to strategies in C' is a Turing machine which, given as input the number of a Turing machine which implements a strategy $s_2 \in C'$, computes the number of a Turing machine for Player 1 which implements some best response s_1 to s_2.[15] The following theorem shows that, for discount factors sufficiently close to 1, no such effective procedure exists.

Theorem 2 *Assume the stage game G satisfies Assumption A. Then there is a $\underline{\delta} > 0$ such that for any rational discount factor $\delta \in (\underline{\delta}, 1)$, there is no effective procedure for computing a best response to strategies in C'.*

Proof. The argument is almost the same as the one given for Theorem 1, so we will be sketchy. Take the strategy used for Player 2 in Theorem 1 and modify it by stripping away all the test periods and reward periods except for the first. Thus the prescribed path now consists of one test period followed by K^r reward periods, followed by adjustment periods which now stretch out forever. Next, form from this modified strategy an infinite set of strategies, which differ only in the "question" asked in the initial test date. Specifically, strategy s_2^n asks whether $n \in A$ or $n \in B$, where A and B are as in the

[15]In fact, an effective procedure for computing a best response is equivalent to an effective procedure which, following any t-period history, computes the action chosen by a best response in period $t + 1$.

proof of Theorem 1. Each of these strategies s_2^n is implementable; moreover, each of these strategies has an implementable best response, since a Turing machine can always be "hard-wired" to play correctly in Period 1.[16]

An effective procedure for computing a best response to this set of strategies would require a Turing machine which, for each n, computes whether $n \in A$ or $n \in B$. Since A and B are recursively enumerable but not recursive, no such Turing machine exists. Hence there is no effective procedure for computing a best response to this set of strategies (and *a fortiori* no effective procedure for computing a best response to the full set C'.) $\square$

[16]Indeed, each of these strategies and a best response is implementable in polynomial time.

References

Dilip Abreu and Ariel Rubinstein (1988), "The Structure of Nash Equilibrium in Repeated Games with Finite Automata," *Econometrica*, **56**, 1259-1281.

Luca Anderlini (1990), "Some Notes on Church's Thesis and The Theory of Games," *Theory and Decision*, **29**, 19-52.

Luca Anderlini and Hamid Sabourian (1993), "Cooperation and Effective Computability,"Cambridge University Working Paper.

Robert Aumann (1981), "Survey of Repeated Games," in *Essays in Game Theory and Mathematical Economics in Honor of Oskar Morgenstern*, Bibliographisches Institut

Elchanan Ben-Porath (1990), "The Complexity of Computing a Best Response Automaton in Repeated Games with Mixed Strategies," *Games and Economic Behavior*, **2**, 1-12.

Ken Binmore (1987), "Modeling Rational Players, Part I," *Economics and Philosophy,* **3**, 179-214.

David Canning (1992), "Rationality, Computability, and Nash Equilibrium," *Econometrica*, **60**, 877-888.

Lance Fornow and Duke Whang (1993), "Optimality and Domination in Repeated Games with Bounded Players," University of Chicago Working Paper.

Itzak Gilboa (1988), "The Complexity of Computing Best Response Automata in Repeated Games", *Journal of Economic Theory*, **45**, 342-352.

Itzak Gilboa and Dov Samet (1989), "Bounded versus Unbounded Rationality: The Tyranny of the Weak," *Games and Economic Behavior*, **1**, 213-221.

John Hopcroft and Jeffrey Ullman (1979), *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, Reading, Massachusetts.

Ehud Kalai and William Stanford (1988), "Finite Rationality and Interpersonal Complexity in Repeated Games," *Econometrica*, **56**, 397-410.

Vicki Knoblauch (1994), "Computable Strategies for Repeated Prisoner's

Dilemma," *Games and Economic Behavior*, **7**, 381-389.

Michael Machtey and Paul Young (1978), *An Introduction to the General Theory of Algorithms*, North Holland, New York.

R. Preston McAfee (1984), "Effective Computability in Economic Decisions," University of Texas Working Paper.

Nimrod Megiddo and Avi Wigderson (1987), "Turing Machines Play the Finitely Repeated Prisoners' Dilemma," Tel Aviv University Working Paper.

John H. Nachbar (1994), "On Computing a Best-Response in a Discounted Supergame," Washington University, St. Louis Working Paper.

John H. Nachbar (1995), "Prediction, Optimization and Rational Learning in Games," Washington University, St. Louis Working Paper.

Piergiorgio Odifreddi (1987), *Classical Recursion Theory*, North Holland, Amsterdam.

Roger Penrose (1989), *The Emperor's New Mind*, Penguin, New York.

Ariel Rubinstein (1986), "Finite Automata Play the Repeated Prisoners' Dilemma," *Journal of Economic Theory*, **39**, 83-96.

William Stanford (1989), "Symmetric Paths and Evolution to Equilibrium in the Discounted Prisoners' Dilemma," *Economics Letters*, **31**, 139-143.