数理解析研究所講究録
1165 巻 2000 年 232-244

KINSHIP-RECOGNITION AND SELF-SACRIFICE IN PRISONERS' DILEMMA

MIKIO NAKAYAMA

Department of Economics, Keio University, 2-15-45 Mita, Tokyo 108-8345.

ABSTRACT. We define the *kinship-recognizing* program (KRP, for short) to play the Prisoners' Dilemma, a generalization of the self-recognizing program discussed by Howard [6], to one that can recognize not only itself but also similar programs as well, thereby admitting mutual cooperation in a wider class of programs. The definition is *self-referential* in that the KRP is a program that can recognize the opponent as the KRP. The existence is proved, under a *recursive* equivalence (kinship) relation, by a *recursion theorem* which is also called a *fixed point theorem* in computability theory. Any KRP is then shown to entail the existence of a program that *sacrifices* itself to the opponents that are kin to the KRP. It is also proved that, under a given kinship relation, no KRP is able to recognize any member of other class of KRP.

1. INTRODUCTION

Theoretical inquiry into how players come to cooperate in a repeated play of the Prisoners' Dilemma has been one of the main subjects in game theory. In the literature, at least two types of reasonings of cooperation can be found: one is the rational cooperation with a punishment mechanism at Nash equilibria, as exhibited in the classical folk theorem and its more recent version; and the other is the cooperation achieved by boundedly rational players. In the latter approach, just as in the pioneering experimental work of Axelrod [3], players are often modeled as machines that act according to predetermined programs. Thus, Abreu and Rubinstein [1] or Neyman [11], for example, have used finite automata as models of boundedly rational players (strategies) in playing the Prisoners' Dilemma. Neyman [11], in particular, has shown that restricting the number of states of automaton players prevents them from counting the stages of repetitions so that cooperating at every stage can be in Nash equilibrium in finite repetitions. Megiddo and

Date: December,1999.

Wigderson [9] showed further that cooperation can be approximated by a Nash equilibrium when players are more sophisticated programs such as Turing machines. Howard [6] has also considered machine players, and argued even more drastically that cooperation is possible in the one shot play of the Prisoners' Dilemma.

The key notion in the argument of Howard [6] is the program that can recognize the opponent as identical to itself. That is, under the environment that players are drawn from a program pool and matched to play the Prisoners' Dilemma, the ability to recognize whether or not the opponent program is identical to itself naturally leads to mutual cooperation, which would be hardly the case between rational human players. Every program is fed as an input the algorithm of the opponent player in the form of the Gödel number, but no player knows its identity; that is, no program is given its own Gödel number separately from the inputs. Thus, the existence of the program, *the self-recognizing program*(*the SRP*, for short), is not a trivial fact. Howard [6] directly constructed the algorithm both by English and by a programming language. Rubinstein [15], in a recent stimulating book, also has presented verbal algorithms of the SRP, and discussed several applications to modeling boundedly rational players.

The purpose of this paper is to treat the SRP in an elementary recursion theoretic setting, thereby generalizing it to a program that is able to recognize not only itself but also other programs which are kin to one another in an appropriate way. We call this program *a kinship-recognizing program* (*KRP*, for short). The generalization is motivated in part by the fact that the SRP cannot, by definition, cooperate even with very slightly different SRPs, implying a sort of inefficiency in the mutual cooperation. We shall define the KRP in a self-referential way to be the program that can recognize the KRP, which will be shown to exist, under a *recursive* equivalence (kinship) relation, by the *second recursion theorem*. We then discuss the play of an KRP and other programs in the Prisoners' Dilemma. It will turn out, in particular, that any KRP entails the existence of a highly altruistic program that *sacrifices* itself to the fellow programs of the KRP, being necessarily exploited by them. Finally, a certain impossibility result will be shown: the ability of KRP cannot be extended to the one deciding whether or not the opponent program is also a member of the set of all KRPs under the same kinship relation.

Though recursion theory or computability theory have recently come to be applied more often than before in the literature of game theory and economic theory, basic concepts and theorems still appear relatively unfamiliar. Therefore, we begin with a brief summary of

computability theory including just the necessary elements in this paper. For formal treatments, the reader may refer to Cutland [5] , or Odifreddi [12].

2. Preliminaries

Let f be a unary *partial* function from $N = \{0, 1, 2, ...\}$ to N. The *domain* of f is the set $Dom(f) := \{x | f(x)\ is\ defined\}$, and the *range* of f is the set $Ran(f) := \{f(x) | x \in Dom(f)\}$. If $Dom(f) = N$, f is called a *total* function. Intuitively, a partial function f is said to be *computable* if there exists a finite algorithm such as a *Turing machine* or a *unlimited register machine* to compute f. The definition is similar for n-ary functions. There are several formalizations of the intuitive concept of *effective computability*, all of which have turned out to be equivalent to the *Turing-machine computability*, giving rise to the well-defined class of all *partial recursive functions*. Thus, the partial recursive functions are considered as the formalization of the functions which are effectively computable in the intuitive sense (*Church's thesis*).

Any algorithm or program computing a unary function is a finite sequence of well-defined instructions. Let $\mathcal{P}$ be a set of all such programs. Then a bijection $\gamma : \mathcal{P} \to G \subset N$ can be defined and is called a *coding* or *Gödel numbering* if γ and γ^{-1} are both computable in the following sense:

(a): Given a particular program $P \in \mathcal{P}$, we can effectively find the code number $\gamma(P) \in G$;

(b): Given a number $n \in G$, we can effectively find the program $P = \gamma^{-1}(n)$.

There are several established ways to code finite objects. Fixing on one coding, every computable (unary) function appears in the enumeration:

$$\varphi_0,\ \varphi_1,\ \varphi_2,\ \varphi_3,\ \cdots$$

where, for each φ_e, the number e is the *index* (code number) of a program computing the function φ_e. Thus, a natural number can be identified with the program with that number as its index.

An n-ary *relation* or *predicate* $R(x_1, \ldots, x_n)$ is said to be *decidable* or *recursive* if its *characteristic function* $c_R(x_1, \ldots, x_n)$ is computable, i.e., if the total function

$$c_R(x_1, \ldots, x_n) = \begin{cases} 1 & if\ R(x_1, \ldots, x_n) \\ 0 & if\ \neg R(x_1, \ldots, x_n) \end{cases}$$

is computable.

Similarly, we say that an n-ary relation $Q(x_1, \ldots, x_n)$ is *partially decidable* if its *partial characteristic function* $f(x_1, \ldots, x_n)$ is computable, i.e., if the partial function

$$f(x_1, \ldots, x_n) = \begin{cases} 1 & if\ Q(x_1, \ldots, x_n), \\ undefined & if\ \neg Q(x_1, \ldots, x_n) \end{cases}$$

is computable. It can be shown that an n-ary relation $Q(x_1, \ldots, x_n)$ is partially decidable iff there is a decidable n+1-ary relation $R(x_1, \ldots, x_n, y)$ such that

$$Q(x_1, \ldots, x_n)\ \ iff\ \ \exists y R(x_1, \ldots, x_n, y).$$

The relation in the right-hand side involves the *unbounded search* for a number y satisfying the decidable relation $R(x_1, \ldots, x_n, y)$. Checking successively for $y = 0, 1, 2, \ldots$ whether or not y satisfies the relation R, the search procedure stops if it finds such a y; otherwise the search goes on for ever. But, if the above search procedure is *bounded*, i.e.,

$$Q(x_1, \ldots, x_n)\ \ iff\ \ \exists y \leq z R(x_1, \ldots, x_n, y),$$

then the relation $Q(x_1, \ldots, x_n)$ becomes decidable, since only a finite number of checking is needed to decide whether or not $R(x_1, \ldots, x_n, y)$. We will use this fact in proving that the kinship relation is decidable.

A subset A of N is said to be *recursive* if the membership relation '$x \in A$' is decidable. Thus, the set of primes, the set of odd numbers, the set N, the empty set and finite sets are immediate examples of recursive sets. It will be easy to see that a finite union of recursive sets are also recursive. Similarly, a subset A of N is called *recursively enumerable* (r.e. for short) if the membership relation '$x \in A$' is partially decidable. Recursive sets are recursively enumerable, since the partial characteristic function for the relation '$x \in A$',where A is recursive, can be always obtained by having the computation of the characteristic function for '$x \in A$' enter a loop whenever $x \notin A$. But, there exists an important r.e. set $\{x | \varphi_x(x)\ is\ defined\}$ that is *not recursive*. This set is at the core of every undecidability result. For example, the well-known *unsolvability of the Halting Problem* '$\varphi_x(y)$ *is defined*' follows from this fact: if the Halting relation '$\varphi_x(y)$ *is defined*' were decidable, so must be the relation '$\varphi_x(x)$ *is defined*', a contradiction. Note that the program x halts in some finite numbers of steps if $\varphi_x(y)$ is defined, so that the Halting Problem is partially decidable. Thus, the Halting problem represents a partially decidable relation that is *not* decidable.

Finally, we list two theorems that will be used to prove our results.

The Second Recursion Theorem.: Let f be a 2-ary computable function. Then, there exists an integer e such that $\varphi_e(x) \simeq f(e, x)$.

Here, the symbol $\simeq$ means that respective values of both sides are either undefined or defined with the same value. The number e is called a *fixed point*; and it can be shown that there are infinitely many fixed points. A fixed point e is the index of a program that computes the function *defined by using e itself*; therefore, it is widely useful in showing the existence of programs defined in a *self-referential* way. Just for this reason, we need this theorem. On the other hand, the next theorem is a source of many impossibility results in computability theory.

The Rice's Theorem.: Suppose that B is a nonempty proper subset of all unary computable functions. Then the problem '$\varphi_e \in B$' is undecidable.

Thus, wheher or not a given function e has a certain non-trivial property is generally undecidable.

3. The Self-Recognizing Program

Let x be the index of a program computing the partial function φ_x. Program x is a player of the Prisoners' Dilemma if the range of the function φ_x is $\{c, d\}$, where we interpret the numbers c and d $(c \neq d)$ as representing cooperation and defection, respectively.

	c	d
c	3,3	0,4
d	4,0	1,1

We will assume that *every program is fed as an input a natural number, the index of the opponent program.* A player x then follows the procedure of its own program: it may decode the input and simulate the behavior of the opponent to determine its output, or may simply ignore it and produce an output, or may produce nothing. We allow the possibility that a player cannot produce an outcome of the game when the opponent program outputs nothing or an irrelevant number. But, the ability of recognition requires a program to compute a total function as follows.

Definition 1. *Program x is said to be a self-recognizing player* (SRP) *if*

$$\varphi_x(y) = \begin{cases} c & if\ x = y \\ d & if\ x \neq y \end{cases}$$

Programs do not *know* their own indices. In other words, no program is fed its own index separately from the index of the opponent program. But, the SRP defined here behaves as if it knew that x is its own index:

it cooperates if the opponent is identical to itself, and defects otherwise. In this sense, we say x is self-recognizing.

Proposition 1. *There exists a self-recognizing player.*

The proof follows from Proposition 2 in the next section, where the existence of a generalized version of an SRP is proved. For an intuitive description of the structure of the SRP, see Howard [6] or Rubinstein [15].

Consider, now, the following program.

Definition 2. *Program z is said to be self-sacrificing if $\varphi_z(z) = d$ and $\varphi_z(y) = c$ for some $y \neq z$.*

An immediate example of a self-sacrificing player would be one that is highly irrational, defecting just against itself and cooperating otherwise. Such a player is seen to exist simply by exchanging c for d in Definition 1. But, a more interesting self-sacrificing player would be the following.

Corollary 1. *Let x be an SRP. Then, there exists a self-sacrificing player $z, z \neq x$ such that*

$$\varphi_z(y) = \begin{cases} c & if\ y = x \\ d & if\ y \neq x \end{cases}$$

Proof. Let z be any program with $z \neq x$ such that $\varphi_z = \varphi_x$. Then, the rest follows from Definition 1. □

The player z might be called $x - altruistic$, sacrificing itself to x and defecting otherwise. It is remarkable that the very existence of an SRP should entail the existence of such an altruistic program in the Prisoners' Dilemma.

4. The Kinship-Recognizing Program

There exist infinitely many SRPs due to the property of the fixed points. As noted by Howard [6], any one of them recognizes no other SRPs no matter how they are similar to itself. For example, no SRP can by definition recognize any program that has just a slightly different syntax yet computes the same function. Any such program is essentially the same to the original SRP, so that they could cooperate with each other if only they had the ability to recognize each other as essentially the same. The SRP is in this sense an inefficient program as far as the mutual cooperation in the Prisoners' Dilemma is concerned.

Let us call program e' a fellow program to e if e' computes the same function as that of e, i.e., $\varphi_{e'} = \varphi_e$. A program might be called a fellow-recognizing program (FRP), if it is a program that can recognize an opponent as a fellow program that can recognize an opponent as a fellow program that can recognize ... *ad infinitum*. This self-referential definition can be substantiated in the following.

For each $e \in N$, let $I_e = \{z|\varphi_z = \varphi_e\}$ be the set of indices of programs that compute the function e. Then, the relation '$y \in I_x$' is an equivalence relation, since the *reflexivity* [$x \in I_x$], the *symmetry* [$y \in I_x$ iff $x \in I_y$] and the *transitivity* [$x \in I_y$ and $y \in I_z$ imply $x \in I_z$] are all satisfied. It will be legitimate, therefore, to require that FRP x recognize any fellow program $y \in I_x$. But, this is a condition asking too much; that is, there is no algorithm to implement the requirement because I_e is not a recursive set.

Formally, this impossibility follows directly from the *Rice's Theorem* by noting that $\emptyset \subsetneq I_e \subsetneq N$. In fact, it is not *recursively enumerable* (see, e.g., Cutland [5, Corollary 6-1.5, p.104; and Exercise 7-2.18(11(d)), p.133]). Intuitively, the non-recursiveness of I_e can be seen by observing that $x \in I_e$ iff $\varphi_x = \varphi_e$ and that the latter relation is not decidable because the equality of functions cannot be assured in any finite number of steps. Thus, there is no algorithm that can decide whether any given two programs are themselves fellows or not.

The only way to circumvent this is to narrow the set of fellow programs down to some recursive subset of I_e by defining a recursive relation to distinguish the opponent program. To this end, we may apply the technique known as *padding*. Specifically, let M be a finite set with $|M| = m \geq 0$ that is a subset of all instructions every one of which is itself redundant and has no influence upon any other instructions. Then, for each index e, consider the program obtained by adding finite numbers of instructions of M with repetitions to the tail of program e. All these programs can be enumerated according to the number and the order of the added instructions as follows:

$$e, e_{11}, \ldots, e_{1m}, e_{21}, \ldots, e_{2m^2}, \ldots, e_{n1}, \ldots, e_{nm^n}, \ldots$$

where e_{nj} is the code of the jth program of all m^n programs with n instructions appended to e. Let this set be represented by

$$I^*_e = \{e_0, e_1, e_2, e_3, \ldots\},$$

where $e_0 = e$. Then, if $m > 1$, the number e_k with $k > 0$ is the index of $j = k - m(m^{n-1} - 1)/(m - 1)$ th program in the m^n programs with $n > 0$ instructions appended to e, i.e., $e_k = e_{nj}$, if and only if $m(m^{n-1} - 1)/(m - 1) < k \leq m(m^n - 1)/(m - 1)$. When $m = 1$, e_k is

just the index of a program with k repetitions of the single instruction appended. The important property of the set I^*_e is the following.

Lemma 1. *I^*_e is a recursive set.*

Proof. Let γ be our coding system. The set I^*_e consists of the indices of all programs of the form

$$J_0 J_1 J_2 \cdots J_r$$

where J_0 is the program P with index e and for $k = 1, \ldots, r$, J_k is the instruction from M added with repetitions just after $k-1$ instructions from M have been appended. Then, we have

$$x \in I^*_e \Leftrightarrow \exists r[x = \gamma(J_0 J_1 J_2 \cdots J_r)].$$

The search for the number r in the right-hand side is *bounded*, since

$$\gamma(J_0 J_1 J_2 \cdots J_r) \geq r, \quad \forall r = 0, 1, 2, \ldots$$

so that

$$x \in I^*_e \Leftrightarrow \exists r \leq x[x = \gamma(J_0 J_1 J_2 \cdots J_r)]$$

Then, the right-hand side is decidable by virtue of the bounded search and the total computability of γ. Hence, the relation '$x \in I^*_e$' is decidable, which implies that the set I^*_e is recursive. □

Intuitively, given x, whether or not x is a member of I^*_e can be decided by first decoding x and then checking if it consists exactly of the instructions identical to the program e with a finite number of instructions just in M being appended. Note that, by construction, the code number e is the minimum in I^*_e; and that I^*_e is an infinite set unless $M = \emptyset$. If we take $M = \emptyset$, we have $I^*_e = \{e\}$.

We now have the recursive relation '$y \in I^*_x$' which, however, is not symmetric. We say x is a *predecessor* of y if $y \in I^*_x$ in that y is derived from x by copying instructions from M. By slightly abusing the use of the word, we say x is a predecessor of itself. A predecessor of x with no predecessor other than itself is called the *ancestor* of x. By construction, for any program x the ancestor of x exists uniquely. We now define an equivalence relation $R^*(x, y)$ from the relation '$y \in I^*_x$' as follows: For all x and y,

$$R^*(x, y) \Leftrightarrow \exists w \textit{ such that } x \in I^*_w \wedge y \in I^*_w .$$

We call this relation R^* a *kinship relation* in that x and y are kin to each other iff they have a predecessor in common. Since the common predecessor also has the ancestor, it follows that x and y are kin to each other iff they share the ancestor in common. The kinship relation is a recursive equivalence relation as shown below, and it reduces to the equality relation when $M = \emptyset$.

Lemma 2. *$R^*(x,y)$ is a recursive, equivalence relation.*

Proof. First, we show that it is an equivalence relation. It will be enough to check the transitivity. Assume that $R^*(x,y)$ and $R^*(y,z)$. Then, there are w and v such that

$$x \in I^*_w \wedge y \in I^*_w, \text{ and } y \in I^*_v \wedge z \in I^*_v.$$

But, then, the ancestors of w and v coincides with that of y, which is then the common ancestor of x and z. Hence, for this ancestor o we have

$$x \in I^*_o \wedge z \in I^*_o,$$

which shows that $R^*(x,z)$, i.e., the transitivity.

To show that R^* is recursive, first note that R^* has a *bounded* search for a number w, that is,

$$R^*(x,y) \Leftrightarrow \exists w \leq z[x \in I^*_w \wedge y \in I^*_w],$$

where $z = \min\{x,y\}$. This must be so, because $0 \leq w \leq x$ and $0 \leq w \leq y$ whenever a common predecessor w of x and y exists. Conjunction of two recursive relations is recursive, and a recursive relation with bounded search is again a recursive relation. Hence, R^* is recursive. □

We are now ready to define the kinship-recognizing program.

Definition 3. *Program x is said to be a kinship-recognizing player* (KRP) *if*

$$\varphi_x(y) = \begin{cases} c & if\ R^*(x,y) \\ d & if\ \neg R^*(x,y) \end{cases}$$

Thus, if x is an KRP and $R^*(x,y)$ with $x \neq y$, then y is also the KRP and $\varphi_x(y) = \varphi_y(x) = c$; that is, a mutual cooperation results, which was impossible for SRPs. Of course, two KRPs x and y such that $\neg R^*(x,y)$, i.e., two KRPs which are not kin to each other defects each other.

Proposition 2. *There exists a kinship-recognizing player.*

Proof. Since the relation $R^*(x,y)$ is recursive, the characteristic function

$$f(x,y) = \begin{cases} 1 & if\ R^*(x,y) \\ 0 & if\ \neg R^*(x,y) \end{cases}$$

is computable. Therefore, the function h defined by

$$h(x,y) = \begin{cases} cf(x,y) & if\ R^*(x,y) \\ d + f(x,y) & if\ \neg R^*(x,y) \end{cases}$$

is computable. The *second recursion theorem* then guarantees the existence of a *fixed point*, i.e., an index x such that $\varphi_x(y) = h(x,y)$. Hence, there exists an index x such that

$$\varphi_x(y) = \begin{cases} c & if\ R^*(x,y) \\ d & if\ \neg R^*(x,y) \end{cases}$$

which implies that program x is kinship-recognizing. □

The SRP x is a special KRP with $R^*(x,y)$ being the recursive relation '$x = y$'.

The fact that the general relation 'x and y compute the same function', i.e., the relation '$y \in I_x$' is not recursive places a limitation on the ability of recognition of KRPs. An KRP x judges an opponent $y \in I_x$ as a stranger unless $R^*(x,y)$. Nevertheless, the bounded ability fares well in the Prisoners' Dilemma. As was the case with an SRP, such an opponent y is a self-sacrificing player that might be called a *kin-to-x-altruistic* player in the following sense.

Corollary 2. *Let x be a KRP. Then, there exists a self-sacrificing player z such that $\neg R^*(x,z)$ and*

$$\varphi_z(y) = \begin{cases} c & if\ R^*(x,y) \\ d & if\ \neg R^*(x,y) \end{cases}$$

The proof follows immediately from Definition 3 by letting $z \in I_x$ satisfy $\neg R^*(x,z)$. The player z sacrifices itself to players that are kin to x, and defects otherwise, which is the extended attribute of the x-altruistic player. Such an altruistic player should exist for *any* recursive equivalence relation $R(x,\cdot)$, since the recursive set $\{y|R(x,y)\}$ is a proper subset of I_x.

If the opponent is not a program computing a total function with range $\{c,d\}$, the outcome of the Prisoners' Dilemma may not be defined so that the game is not playable. Even when the opponent is a player of the Prisoners' Dilemma, this can happen since players are programs computing partial functions. In the next section, we will discuss some undecidability results concerning the type of the opponent program.

5. The Undecidability of KRP

The ability of an KRP to cooperate in the Prisoners' Dilemma is considerably improved from that of an SRP. The domain of cooperation is not a singleton set, but an infinite recursive subset of N. Yet, just like the SRP, an KRP x does not cooperate across the different classes of kinship since $\varphi_x(y) = d$ for *any* program y unless $R^*(x, y)$: it cannot decide whether y is an KRP at all or not when $\neg R^*(x, y)$. We now show that this is a fundamental limitation placed on all programs in general.

Let $R(x, y)$ be any recursive equivalence relation, and let $K(R)$ be the set of all indices of KRPs under R. If there existed an algorithm to decide the membership in $K(R)$, then such an algorithm could be used to define the KRP x recognizing any KRP y under R as follows:

$$\varphi_x(y) = \begin{cases} c & if\ x \in K(R) \wedge y \in K(R) \\ d & if\ x \notin K(R) \vee y \notin K(R) \end{cases}$$

Note that the relation $x \in K(R) \wedge y \in K(R)$ is an equivalence relation. Thus, the domain of cooperation could be extended to $K(R)$. But, here again the *Rice's Theorem* stands in the way, since $\emptyset \subsetneq K(R) \subsetneq N$.

Proposition 3. *The set $K(R)$ of all indices of* KRPs *under any recursive equivalence relation R is not a recursive set.*

Behind this undecidability is an intuition that, given any index y, the decision procedure needs infinitely many steps to assure the program y being able to distinguish its kin from others for each of the infinite numbers of opponent programs.

Technically, more is involved in the above undecidability result: the set $K(R)$ is not recursively enumerable, which follows from the theorem of Rice and Shapiro (see, Cutland [5, Theorem 7-2.16, p.130]); and the complement $K(R)^c$ in N is also not recursively enumerable (Cutland [5, Theorem 7-3.4, p.135]). This means that the decision problem whether or not y is an KRP is not just undecidable in the same sense as the *Halting problem*, but far more difficult to solve than the Halting Problem.

This undecidability result is not so surprising, however. This is but one example of many undecidability results obtainable by computability theory when applied to game theory [see, e.g., Binmore [4] or Anderlini [2] for fundamental restrictions placed on rational program players; Prasad [13] for undecidability of the existence of Nash equilibria in infinite games; Knoblauch [8] or Nachbar and Zame [10] for non-computability of some repeated- game strategies; and, Jones [7] or

Rabin [14] for more classical undecidability results of certain perfect-information win-lose games.

The one shot play of the Prisoners' Dilemma as considered in this paper is subject to a more primitive undecidability of the type discussed by Anderlini [2], which are now seen to follow directly from the Rice's Theorem. For example, no program can, given any opponent program, decide whether or not the opponent is a player of the Prisoners' Dilemma; namely, the set $I := \{e|Ran(\varphi_e) = \{c,d\}\}$ is not recursive because $\emptyset \subsetneq I \subsetneq N$. Similarly, there is no program to decide if the game to be played can be played at all, for the set $I^\circ := I \cap \{e|\varphi_e \textit{ is total}\}$ is not recursive. In other words, whether or not a given program is a player capable of always reaching a decision in the Prisoners' Dilemma is not decidable; in fact, it is very badly undecidable just as the membership relation to the set K of all KRPs. Thus, as far as the playability of the game is concerned, an outside, intelligent umpire (like an *oracle* in computability theory) will be necessary. The KRP could always play the Prisoners' Dilemma only when the opponent programs were drawn from the non-recursive set P of programs computing total functions with range $\{c,d\}$.

References

[1] Abreu, D., and A. Rubinstein, "The Structure of Nash Equilibrium in Repeated Games with Finite Automata," *Econometrica*, 56 (1988), 1259–1281.

[2] Anderlini, L., "Some Notes on Church's Thesis and the Theory of Games," *Theory and Decision*, 29 (1990), 15–52.

[3] Axelrod, R., *The Evolution of Cooperation*, Basic Books, New York 1984.

[4] Binmore, K. G., "Modeling Rational Players, Part I," *Economics and Philosophy* 3 (1987), 179–214.

[5] Cutland, N. J., *Computability*, Cambridge University Press, Cambridge, 1980.

[6] Howard, J. V., "Cooperation in the Prisoners' Dilemma," *Theory and Decision*, 24 (1988), 203–213.

[7] Jones, J. P., "Some Undecidable Determined Games," *International Journal of Game Theory* 11 (1982), 63–70.

[8] Knoblauch, V., "Computable Strategies for Repeated Prisoners' Dilemma," *Games and Economic Behavior*, 7 (1994), 381–389.

[9] Megiddo,N and A.Wigderson, "On Play by Means of Computing Machines," in *Theoretical Aspects of Reasoning About Knowledge*, Poceedings of the 1986 Conference, J.Y.Halpern ed. Los Altos:Kaufmann, 1986.

[10] Nachbar, J.H. and W.R. Zame, "Non-Ccomputable Strategies and Discounted Repeated Games," *Economic Theory*, 8 (1996), 103–122.

[11] Neyman, A., "Bounded Complexity Justifies Co-Operation in the Finitely Repeated Prisoners' Dilemma," *Economics Letters*, 19 (1985), 227–229.

[12] Odifreddi, P., *Classical Recursion Theory*, North-Holland, Amsterdam, 1992.

[13] Prasad, K., "Computability and Eandomness of Nash Equilibrium in Infinite Games," *Journal of Mathematical Economics*, 20 (1991), 429–442.

[14] Rabin, M.O., "Effective Computability of Winning Strategies," in *Contributions to the Theory of Games, Annals of Math. Studies*, 39, M.Dresher, et.al eds., Princeton University Press,1957.

[15] Rubinstein, A., *Modeling Bounded Rationality*, MIT Press, Cambridge, Massachusetts, 1998.