E C O N O M I C S B U L L E T I N

Can A Turing Player Identify Itself?

David Levine
UCLA

Balázs Szentes
University of Chicago

Abstract

We show that the problem of whether two Turing Machines are functionally equivalent is undecidable and explain why this is signif− icant for the theory of repeated play and evolution.

We thank National Science Foundation Grants SES−03−14713 and SES−05−18762 for financial support.
Citation: Levine, David and Balázs Szentes, (2006) "Can A Turing Player Identify Itself?." *Economics Bulletin*, Vol. 1, No. 1 pp. 1−6
Submitted: January 29, 2006. **Accepted:** March 23, 2006.
URL: http://www.economicsbulletin.com/2006/volume1/EB−06A00001A.pdf

1. INTRODUCTION

The wide empirical prevalence of cooperation in prisoner's dilemma type situations where free-riding is possible has long fascinated economists, game theorists and biologists. The possibility of punishing free-riding in repeated situations is central to the theory of repeated games, but the folk-theorem leaves open the possibility of coordination failure and Pareto inferior equilibria. A key question addressed in the evolutionary literature is whether cooperation is merely possible, or whether in fact there are evolutionary forces that push in the direction of cooperation. There are a number of affirmative answers to the question: central to evolutionary forces that lead to cooperative outcomes is the ability to identify free-riders so they can be punished.[1]

There are three basic mechanisms through which "deviant" free-rider strategies can be detected: indirectly by observing behavior, through some sort of "cheap talk" in the form of a "secret handshake," or directly by observing the strategies themselves. The "secret handshake" is a signal that is secret from free-riders and is discussed by Robson [1990]. In the setting of play between finite state machines "secret handshakes" take place through observing play as described in Rubinstein [1986], for example. The possibility of directly observing strategies is examined in Levine and Pesendorfer [2002] - they show a strong result that the only strategies to survive a stochastic process of evolution and imitation are strategies that cooperate when they meet themselves, and punish opponents who use different strategies.

Here we take up the issue of directly observing the strategies used by opponents. In other words, we envisage a setting where players exchange descriptions of their strategies prior to play. There are several reasons why this is interesting. First, whether information is generated through cheap-talk or through observation of behavior, it is not possible to have more information about an opponents intentions than through direct observation of her strategy. So direct observation gives an upper bound on how well a player can be informed about an opponent's intentions - any limitations when direction observation is used must also be limitations when intentions must be inferred indirectly. Second, there are interesting circumstances where direct observation of strategies is possible. For example, in a biological setting, strategies are encoded in DNA, and direct detection of DNA or its attributes is possible. A good example of this is the immune system which is based on detecting "free-riders" through their chemical attributes rather than their behavior. In the economic setting, institutions that make it easy to observe opponents strategic rules - for example, implementation by third parties, or the ability to credibly reveal the rules that are being used through institutions such as bureaucracies - are possible and indeed frequent.

Technically, a strategy is simply an algorithm that responds to certain inputs. Computer scientists conjecture that all algorithms that can be implemented and that meet certain minimal requirements are Turing machines,[2] so this is a natural way in which to discuss strategic algorithms.[3] Economists such as Binmore [1990] and Anderlini [1990] have explicitly modeled player strategies as Turing Machines: they have focused on showing that there does not exist a machine which

[1] In the literature on positive assortive matching, discussed, for example in Berstrom [2002], the matching mechanism implicitly identifies free-riders by matching players based on their play. This can be interpreted as indirectly detection through observed behavior.

[2] This is known as Church's Thesis.

[3] An alternative would be to consider finite state machines, also know as automata, a subclass of Turing Machines with finite memory. This would lead to a different set of issues: how much success could we expect a machine with n states to have in identifying the strategy of a machine with some larger number $m > n$ states?

always gives a best-response to the opponent's strategy.[4] The issues can be illustrated through the example of matching pennies. Suppose that each player designs a Turing machine that takes as input the Turing machine designed by the other player. An obvious strategy would be to simulate the opponent's Turing machine, using your own Turing machine as input. The problem with this is that when both players use such Turing machines, we get an infinite regress in which I simulate you simulating me simulating you...and there is an infinite loop. In technical terms, these Turing machine do not halt when given each other as input, and so are useless in practice.

In the setting of punishing free-riders, the situation is less difficult. It is easy to design a machine that when given opposing Turing machine as input checks to see if that machine is the same or different.[5] The strategy is then to cooperate if the opposing machine is the same and punish if it is different. This is the basis for the Pesendorfer-Levine [2002] result, for example. However, the same strategy may be described by many different Turing machines. For example, one machine might engage in some irrelevant preliminary calculations that are later discarded, while another does not. This raises the question whether it is possible to identify an opponent not by the details of how their strategy is described, but according to the strategy or function that their algorithm implements. That is the question addressed in this note. It is important for three reasons. First, the interesting question about opponent play is how they will behave, not how their code is written. The question we would like to know from examining an opponent's strategy are of the type: Is my opponent going to cheat, so I should do likewise? Is he going to engage in a strategy that leaves him open to exploitation? Second, there is an obvious social disadvantage if machines that differ merely in external appearance engage in costly punishment against one another. Third, if strategies can only be determined through observing behavior, then it is evident that only the function of a machine can be identified.

The questions we will examine, then, is the extent to which it is possible to identify an algorithm based on its function, rather than on its description.[6] We are not going to require Turing Machines to predict the strategies of their opponents, but focus on the narrower question at issue in sustaining cooperation: can Turing Machine recognize whether their opponents are computationally equivalent to themselves?

2. The Question

We let τ denote a Turing Machine, and $\tau(x)$ denote the output of τ on input the integer x,[7] where $\tau(x)$ is an integer, or $\tau(x) = \infty$ if τ does not halt on input x. By convention, since there are countably many Turing Machines, we identify their descriptions with integers. We let $\langle\tau\rangle$ denote the description of the Turing Machine τ. In other words $\tau'(\langle\tau\rangle)$ is the output of the Turing Machine τ' when the input is the description of the Turing Machine τ. If we have two Turing Machines τ_1 and τ_2 we say that they are *equivalent* if they compute the same function, that is $\tau_1(x) = \tau_2(x)$ for all x. We say that a Turing Machine is *finite* if it halts on every input. A problem with a yes/no

[4]Canning [1992] shows that if the domain of possible players and games are appropriately restricted then Turing Machines will play Nash Equilibria. These restrictions are fairly severe.

[5]This is obvious for Turing Machines. Since a program that checks an input to see if it is identical to itself requires only enough memory to store itself and the input, it is also true for finite state machines.

[6]An alternative approach is to consider Bayesian priors as in Nachbar [1997].

[7]Occasionally we refer to Turing machines that take as input pairs of integers; since there is a computable 1-1 map from pairs of integers to integers, such a machine can be understood as taking as input the single integer generated by a fixed map of this type.

answer is *decidable* if there is a Turing machine that provides the output zero if the answer is no and the output one if the answer is yes.

We consider a series of questions about the decidability of the functional equivalence of two Turing machines. In each case, we ask if there is a Turing machine that can determine whether two Turing machines are computationally equivalent. The answer to each question is no: the problem is not decidable, or in other words, no Turing machine can determine equivalence in any of these cases.

Question 1: Is it a decidable problem whether two Turing Machines are equivalent?

Question 2: Is it a decidable problem whether two finite Turing Machines are equivalent?

Question 3: Is there some Turing Machine τ for which it is a decidable problem whether a Turing Machine is equivalent to τ?

Question 4: Is there some finite Turing Machine τ for which it is a decidable problem whether another finite Turing Machine is equivalent to τ?

We should emphasize that the problem of functional equivalence of Turing machines has not escaped the notice of computer scientists. For example, Question 1 is essentially a restatement of Rice's Theorem.[8] Since the proofs are short, and the terminology of "index sets" used in Rice's Theorem may not be familiar, we give simple direct proofs that the answers to all of these questions is "no."

3. The Answer

Our point of departure is the famous Halting Lemma that determining whether or not a Turing Machine halts is not a decidable problem - that is, there is no Turing Machine which can take as input the description of a Turing Machine and an integer n and compute 1 if it halts on n and 0 if it does not. In other words, whether $\tau(\langle\tau\rangle)$ halts or not is not decidable. For expositional purposes we provide a proof for this result from which the answers to Questions 1-4 will follow.

Halting Lemma. *Whether a Turing Machine halts on itself is undecidable.*

Proof. Suppose by contradiction, that there exists a Turing Machine τ^* such that $\tau^*(\langle\tau\rangle) = 1$ if $\tau(\langle\tau\rangle)$ halts and zero otherwise. Construct the Turing Machine τ' as follows.

$$\tau'(\langle\tau\rangle) = \begin{cases} 0 & \text{if } \tau^*(\langle\tau\rangle) = 0 \\ \infty & \text{if } \tau^*(\langle\tau\rangle) = 1. \end{cases}$$

Then $\tau'(\langle\tau'\rangle) = 0 \Leftrightarrow \tau^*(\langle\tau'\rangle) = 0$ by the construction of τ'. However, by the definition of τ^*, $\tau^*(\langle\tau'\rangle) = 0$ is true if and only if τ' does not halt, that is, $\tau'(\langle\tau'\rangle) = \infty$. So $\tau'(\langle\tau'\rangle) = 0 \Leftrightarrow \tau'(\langle\tau'\rangle) = \infty$ an obvious contradiction. □

We now answer each question in the negative by showing that if it had a positive answer then we could find a Turing Machine that determines whether $\tau(\langle\tau\rangle)$ halts for any Turing Machine τ, contradicting the fact that this problem is undecidable.

In addition to the Halting Lemma, the other key ingredient in our proofs will be the idea of one Turing machine simulating another. That is, if we are given as input the description of a Turing machine $\langle\tau\rangle$ and an input x we can as part of our design simply execute $\tau(x)$ to see what the input Turing machine τ would do on the input x. Of course, $\tau(x)$ might not halt, in which case the machine we are designing would not halt either.

[8] See Corollary 6.1.5 in Cutland [1980].

Answer to Question 1: *Is it a decidable problem whether two Turing Machines are equivalent?* No. Fix a Turing Machine τ. Define τ' as follows. If $x \neq \langle\tau\rangle$ then $\tau'(x) = \tau(x)$. If $x = \langle\tau\rangle$ then $\tau'(\langle\tau\rangle) = \infty$. To construct τ' is easy. It is almost identical to τ, except that first it checks whether the input is $\langle\tau\rangle$ or not. If it is, it runs forever, otherwise it simulates τ. Notice, that in order to define τ' one does not need to know whether $\tau(\langle\tau\rangle)$ halts or not. Observe, that τ and τ' are equivalent if and only if $\tau(\langle\tau\rangle)$ does not halt. Hence, if there was a Turing Machine which determines whether τ and τ' are equivalent, it could also determine whether $\tau(\langle\tau\rangle)$ halts or not.

Answer to Question 2: *Is it a decidable problem whether two finite Turing Machines are equivalent?* No. Fix a Turing Machine τ. Define the Turing Machine $\widehat{\tau}$ as follows: $\widehat{\tau}(n) = 1$ if $\tau(\langle\tau\rangle)$ did not halt after n steps, and zero otherwise. Let $\widetilde{\tau}$ be a Turing Machine such that $\widetilde{\tau}(n) = 1$ for all n. Notice that $\widehat{\tau}$ and $\widetilde{\tau}$ are finite, and in addition they are equivalent if and only if $\tau(\langle\tau\rangle)$ does not halt. Hence a machine that could determine if two finite machines are equivalent could determine if $\tau(\langle\tau\rangle)$ halts.

Answer to Question 3: *Is there some Turing Machine τ for which it is a decidable problem whether a Turing Machine is equivalent to τ?* No. Before we start to answer the question, note that there is a Turing Machine τ_s such that if τ halts on at least one input, then $\tau_s(\langle\tau\rangle)$ is an input on which τ halts. The way to construct τ_s is the following. First τ_s simulates the first step of τ on 1. Then if τ_s simulated the nth step of τ on k, for $n \neq 1$, it will simulate the $n-1$th step of τ on $k+1$, while for $n = 1$ it will simulate the $(k+1)th$ step on input 1. In other words

step	input
1	1
2	1
1	2
3	1
2	2
1	3
4	1

and so forth. Once τ_s finds a step where τ halts, it stops and its output is the input of τ on which it halted.

Let τ be some arbitrary Turing Machine, and let τ_∞ denote a Turing Machine which does not halt on any input. Suppose by contradiction that there exists a Turing Machine $\widehat{\tau}$ such that it is decidable whether or not $\widehat{\tau}$ is equivalent to another Turing Machine. This would mean that there exists a Turing Machine τ^* such that $\tau^*(\langle\tau'\rangle) = 1$ if τ' is equivalent to $\widehat{\tau}$ and zero otherwise.

We now construct a Turing Machine $\widetilde{\tau}$ as follows. The construction depends on which of the following two cases holds.

Case 1: $\tau^*(\langle\tau_\infty\rangle) = 1$. Then define $\widetilde{\tau}$ as follows. If $x \neq 1$, $\widetilde{\tau}(x) = \widehat{\tau}(x)$. If $x = 1$, then $\widetilde{\tau}$ simulates τ on $\langle\tau\rangle$. Thus $\widetilde{\tau}(1)$ halts if and only if $\tau(\langle\tau\rangle)$ does.

Case 2: $\tau^*(\langle\tau_\infty\rangle) = 0$. Since this means that $\widehat{\tau}$ is not equivalent to τ_∞, it implies that $\widehat{\tau}$ halts for some input. Hence $\tau_s(\langle\widehat{\tau}\rangle)$ is well-defined. This enables us to define $\widetilde{\tau}(x) = \widehat{\tau}(x)$ whenever $x \neq \tau_s(\langle\widehat{\tau}\rangle)$. If $x = \tau_s(\langle\widehat{\tau}\rangle)$, then

- *First:* $\widetilde{\tau}$ simulates τ on $\langle\tau\rangle$. If it halts then
- *Second:* $\widetilde{\tau}$ simulates $\widehat{\tau}$ on $x = \tau_s(\langle\widehat{\tau}\rangle)$ and provides the same output as $\widehat{\tau}$.

So $\widetilde{\tau}(\tau_s(\langle\widehat{\tau}\rangle))$ halts if and only if $\widetilde{\tau}$ is functionally equivalent to $\widehat{\tau}$.

We conclude that $\tau(\langle\tau\rangle)$ halts if and only if $(\tau^*(\tau_\infty), \tau^*(\widetilde{\tau})) \in \{(1,0),(0,1)\}$. So τ^* can determine if $\tau(\langle\tau\rangle)$ halts.

Answer to Question 4: *Is there some finite Turing Machine τ for which it is a decidable problem whether another finite Turing Machine is equivalent to τ?* No. Suppose by contradiction that there exists a finite Turing Machine $\widehat{\tau}$ such that it is decidable whether or not $\widehat{\tau}$ is equivalent to another finite Turing Machine. Fix a Turing Machine τ and construct a Turing Machine $\widetilde{\tau}$ as follows. On input n, $\widetilde{\tau}$ simulates the first n steps of τ on $\langle\tau\rangle$. Then $\widetilde{\tau}$ computes $\widehat{\tau}(n)$. If τ halts before the nth step on $\langle\tau\rangle$ then $\widetilde{\tau}(n) \neq \widehat{\tau}(n)$, otherwise $\widetilde{\tau}(n) = \widehat{\tau}(n)$. Obviously, $\widetilde{\tau}$ is finite because $\widehat{\tau}$ is also finite. Furthermore, $\widetilde{\tau}$ and $\widehat{\tau}$ are equivalent if and only if $\tau(\langle\tau\rangle)$ does not halt.

REFERENCES

[1] Anderlini, L. [1990], "Some Notes on Church's Thesis and the Theory of Games," *Theory and Decision* **29**, 19-52.

[2] Bergstrom, T. [2002], "Evolution of Social Behavior: Individual and Group Selection," *The Journal of Economic Perspectives* **16**, 67-88.

[3] Binmore, K. [1990], *Essays on the Foundation of Game Theory.* Oxford: Basil Blackwell.

[4] Canning, D. [1992], "Rationality, Computability, and Nash Equilibrium," *Econometrica* **60**, 877-888.

[5] Cutland, N. J. [1980], *Computability: An introduction to Recursive Function Theory*, Cambridge University Press.

[6] Levine, D. K. and W. Pesendorfer [2002], "Evolution of Cooperation Through Imitation," UCLA.

[7] Nachbar, J. [1997], "Prediction, Optimization, and Learning in Repeated Games," *Econometrica* **65**, 275-309.

[8] Robson, A. J. [1990], "Efficiency in evolutionary games: Darwin, Nash and the secret handshake," *Journal of Theoretical Biology* **144**, 379-96.

[9] Rubinstein A. [1986], "Finite Automata Play the Repeated Prisoner's Dilemma," *Journal of Economic Theory* **39**, 83-96.