CAN A TURING PLAYER IDENTIFY ITSELF?

DAVID K. LEVINE AND AND BALÁZS SZENTES

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

Keywords: Economic Theory, Game Theory.
JEL Classification: A1; A2

1. INTRODUCTION

Consider the basic problem of cooperation in a Prisoner's dilemma type situation. Cooperative outcomes must be enforced by punishing free-riders - to do so it is necessary to identify free-riders so that they can be punished. This observation is a feature of the literature on repeated play with bounded rationality and in the theory of the evolution of cooperation. In both cases, a "good" strategy is to give some sort of "secret handshake" to determine if the opponent is of the same type - and so should be cooperated with - or a different type that should be punished. In the setting of play between finite state machines, the "handshake" is described in Rubinstein [1986]; in the evolutionary setting by Robson [1990]. Typically, these "secret handshakes" take the form of some sort of signalling during the course of play, but any strategy must be generated by an algorithm, and in some circumstances it may be able to inspect the opponents algorithm prior to play - this has obvious advantages over testing the opponent during the course of play. An explicit theory along these lines can be found in Levine and Pesendorfer [2002]. Of course an algorithm may be described in many ways - from a strategic perspective, what is important is not details of the algorithm, but rather its functionality. 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?

Date: First Version: 20th January 2006, This Version: 16th March 2006.

UCLA and The University of Chicago. Corresponding Author - David K Levine: Department of Economics, UCLA, Los Angeles, CA 90095, USA. Phone/Fax: 310-825-3810. Email: david@dklevine.com. Balász Szentes: Department of Economics, The University of Chicago, 1126 59th Street, Chicago, IL 60637. Email: szentes@uchicago.edu.

We thank National Science Foundation Grants SES-03-14713 and SES-05-18762 for financial support.

The purpose of this note is to examine the extent to which it is possible to identify an algorithm based on its function, rather than on its description.[1] A natural way to describe algorithms is as a Turing Machine. Binmore [1990] and Anderlini [1990] were among the first to model players as Turing Machines. In their models, machines receive a description of their opponent and the game played as inputs. The authors show that there does not exists a machine which always gives the best response to the opponent's strategy.[2] In this note we do not require Turing Machines to predict the strategies of their opponents. We merely want to know if the strategy of a Turing Machine can determine whether its opponent is equivalent to itself. In other words: Are there Turing machines that can 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 x, where $\tau(x) = \infty$ if τ doesn't halt on 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.

We consider the following questions, and prove that the answer is no to all of them.

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?

Recall that for example the problem posed in Question 1 is said to be decidable if there is a Turing Machine τ^* such that $\tau^*(\langle\tau\rangle, \langle\tau'\rangle) = 1$ if τ and τ' are equivalent and 0 if they are not.

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

[2] 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.

3. THE ANSWER

Our point of departure is the basic result 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. More precisely, from the proof of the Halting Lemma whether $\tau(\langle\tau\rangle)$ halts or not is not decidable. For expositional purposes we provide a proof for this result.

Halting Lemma. *Whether a Turing Machine halts on itself is undecidable.*[3]

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.

Answer to Question 1: 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: 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.

[3] An alternative line of proof is to use Rice's Theorem. Like the Halting Lemma, this is a basic result in computability proven in any text on the subject. In fact Question 1 is essentially a restatement of Rice's Theorem. However for the benefit reader unfamiliar with the terminology of "index sets" used in Rice's Theorem we give simple direct proofs of all of the results.

Answer to Question 3: 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.

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: 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 Chruch's Thesis and the Theory of Games," *Theory and Decision* 29, 19-52.

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

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

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

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

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

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