NAG 2-593

Undecidability in Macroeconomics (Preliminary Draft)*

IN-83-CR

Siddharth Chandra**
Tushar Deepak Chandra***

160267
P-40

TR 93-1340
April 1993

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Research supported by NSF grant CCR-9102231, DARPA/NASA Ames grant NAG 2-593, grants from the IBM Endicott Programming Laboratory and Siemens Corp.
**Department of Economics, Uris HAll, Cornell University, Ithaca, NY 14853. This author is also supported by a Cornell Sage fellowship.
***Department of Computer Science, Upson Hall, Cornell University, Ithaca, NY 14853. This author is also supported by an IBM graduate fellowship.

Undecidability in Macroeconomics (Preliminary Draft)*

Siddharth Chandra† and Tushar Deepak Chandra‡

April 28, 1993

Abstract

In this paper we study the difficulty of solving problems in economics. For this purpose, we adopt the notion of *undecidability* from *recursion theory*. We show that certain problems in economics are undecidable, i.e., cannot be solved by a *Turing Machine*, a device that is at least as powerful as any computational device that can be constructed [2]. In particular, we prove that even in *finite closed economies* subject to a variable initial condition, in which a social planner knows the behavior of every agent in the economy, certain important social planning problems are undecidable. Thus, it may be impossible to make effective policy decisions.

Philosophically, this result formally brings into question the *Rational Expectations Hypothesis* which assumes that each agent is able to determine what it should do if it wishes to maximize its utility. We show that even when an optimal rational forecast exists for each agent (based on the information currently available to it), agents may lack the ability to make these forecasts. For example, Lucas [7] describes economic models as "mechanical, artificial world(s), populated by .... interacting robots". Since any mechanical robot can be at most as computationally powerful as a Turing Machine, such economies are vulnerable to the phenomenon of undecidability.

*Research supported by NSF grant CCR-9102231, DARPA/NASA Ames grant NAG-2-593, grants from the IBM Endicott Programming Laboratory and Siemens Corp.

†Department of Economics, Uris Hall, Cornell University, Ithaca, NY 14853. This author is also supported by a Cornell Sage fellowship.

‡Department of Computer Science, Upson Hall, Cornell University, Ithaca, NY 14853. This author is also supported by an IBM graduate fellowship.

Contents

1 Introduction

In this paper we prove that there are fragments of macroeconomics about which it is not possible to reason. In order to do so, we adopt techniques from recursion theory that are new to the study of macroeconomics. There are two implications of this result. First, agents in such economies cannot reason about potentially important facts. Second, economists studying these economies cannot reason about them either. This result prompts us to question the applicability of various traditional assumptions about knowledge in economics. These include but are not restricted to the Rational Expectations Hypothesis and a variety of systems involving the formation of subjective or objective probability priors. We emphasize that this result applies to *closed finite economies.* By extension, we also provide an insight into what "...a formal theory of rational decision-making in an *open*[1] universe..."[1] might confront.

This result does not apply to the kinds of models macroeconomists are used to seeing in journals. Those models have been designed to provide neat answers to a variety of questions. An important feature that is almost invariably lacking in such models, however, is a sense of separation of numerous heterogeneous agents or sectors. The result of this paper might explain why economic models which mimic economies as distributed systems of interacting agents or sectors are not very popular today: it is difficult and sometimes impossible to reason about such economies.

Discrete decision processes are central to economic research. Examples of these processes are the Overlapping Generations Model [10, 15] and the Townsend Turnpike Model [10, 13]. It is natural to ask whether a systematic method solves such optimization problems for economies with heterogeneous agents. This question is extremely difficult to analyze. So we address the simpler question: given an arbitrary economy for which there is a solution to a decision making problem, is it possible to find that solution? In this paper, we show by example that this may not be possible *even for a finite economy.* Thus the problem is "hard" because of its inherent complexity, and not because the economy itself is intractably large. We present a problem that is specified in all respects except for its initial condition and show that it is "hard" to solve.

To prove such a result we use the concept of a *Turing Machine.* A Turing Machine is a computing device which is capable of executing any possible computing algorithm including every possible learning algorithm and fine tuning mechanism. There is strong evidence that a Turing Machine is at least as powerful as any computational device that can be constructed [2]. A problem that cannot be solved by a Turing Machine is said to be *undecidable.* We use techniques introduced by Reif et al [9] to develop a methodology for showing that a variety of problems in economics are undecidable. We apply this methodology to a simple problem for which a solution exists, and show that the solution cannot be found using a Turing Machine. We thus demonstrate the degree of difficulty that economists would encounter in characterizing a network economy in terms of discrete interacting agents[2] with limited choices.

[1] The italics are ours.

[2] These could be consumers, sectors, producers, trading partners etc.

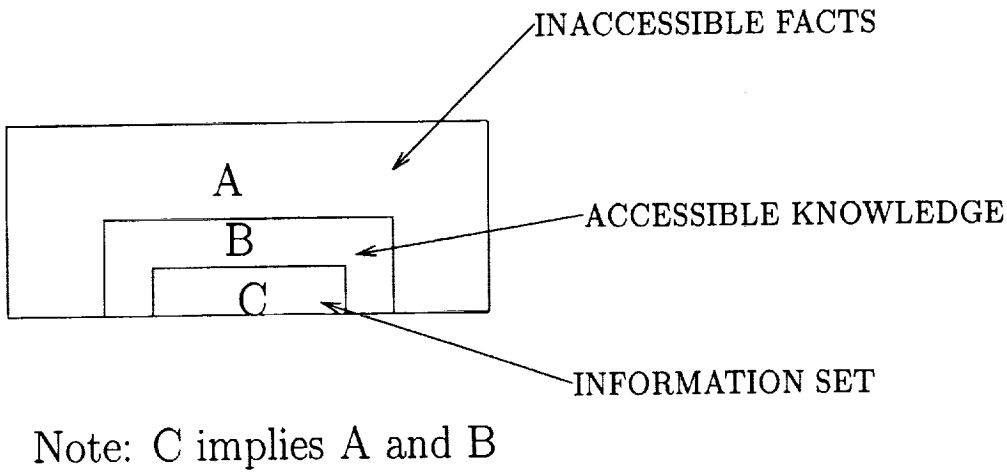

Figure 1: We prove the existence of inaccessible facts.

This paper is divided into two parts. In the first part we present a discussion of knowledge in economics and an example of an economy in which there are undecidable problems. The second part consists of four appendices, the last two of which introduce optimization and recursion theory to readers without any background in these subjects. We strongly recommend that readers who are unfamiliar with the economic or computational concepts used in the first part of the paper refer to these appendices. Supplementary remedial readings are also suggested in each appendix.

1.1 Rational Expectations: a limitation

We use the exposition in Wan [16] to summarize rational expectations:

- Individuals form subjective probability distributions over future events.
- These distributions agree with the objective probability distribution of events based on given information.
- This information includes past history, government policy and the set of relations and functions that make up the general environment.
- Based on this information, a known outcome follows.

Thus, in order for rational expectations to hold, facts that follow from and only from an agent's information must be available to the agent. In this paper, we show that there exist facts that follow from and only from an agent's information that are not computable by that agent even if (s)he knows the complete specification of the economy (see figure 1). Rational expectations as it is widely understood today shows no awareness of this

phenomenon. While this result is philosophical in nature, it should be of concern to a variety of economists. In section 3 we outline a few examples in which undecidability is a concern. The work of Spear [12] is related to this subject.

2 The Problem

2.1 Introduction

In this section, we will state a problem for a finite closed network economy and show that it is undecidable. Intuitively, our construction will proceed as follows. Suppose we are given a Turing Machine specification x and an input i. We will construct an economy $\mathcal{E}(x)$ with an agent α, and an initial impulse $I(i)$ with the following property. The impulse $I(i)$ reaches α in $\mathcal{E}(x)$ if and only if ϕ_x halts on i. In other words, we will *reduce* the halting problem into a decision problem for a finite closed economy. By doing so, we will have shown that the set of such problems is not recursive. This follows from the fact that the halting problem is not recursive.

2.2 Encoding the Turing Machine

Suppose the given Turing Machine is $\mathcal{T} = \langle Q_{\mathcal{T}}, \Gamma, \perp, \Sigma, \delta_{\mathcal{T}}, q_0, q_a, q_r \rangle$ where $\Gamma = \{0, 1\}$.[3] We construct a new Turing Machine $\mathcal{M}$ that has exactly one halting state q_h such that $\mathcal{M}$ halts on exactly the same set of inputs as $\mathcal{T}$.

The set of states of $\mathcal{M}$ is $Q = Q_{\mathcal{T}} \cup \{q_h\}$. The state transition function of $\mathcal{M}$, denoted δ, includes all the transitions of $\delta_{\mathcal{T}}$ and the following four additional transitions:

$$\delta(q_a, 0) = \delta(q_a, 1) = \delta(q_r, 0) = \delta(q_r, 1) = \langle q_h, 1, R \rangle$$

The reader may verify that the resulting Turing Machine $\mathcal{M}$ halts on exactly the same set of inputs as $\mathcal{T}$. We will now construct an economy that mimics the behavior of $\mathcal{M}$ and therefore mimics $\mathcal{T}$.

Our economy consists of n agents, where n denotes the number of states in Q. Each agent corresponds to a unique state of $\mathcal{M}$. We represent the storage tape of $\mathcal{M}$ by using two binary fractions a_s and b_s. Let g_0 be the symbol which the tape head is currently scanning.

Let $g_1, g_2, g_3, \ldots$ be the successive symbols on the left of g_0, and $h_0, h_1, h_2, \ldots$ be the successive symbols on the right of g_0. This is shown in figure 2. Then, we can represent the storage tape by using two numbers a_s and b_s:[4]

$$a_s = \sum_{i=0}^{\infty} \frac{g_i}{3^{i+1}} \tag{1}$$

$$b_s = \sum_{i=0}^{\infty} \frac{h_i}{3^{i+1}} \tag{2}$$

Turing Machine moves can be divided into two cases, left moves and right moves. Thus we consider two cases of δ:

[3] This assumption on Γ does not reduce the computational power of the Turing Machine.

[4] We use 3 in the denominator of our expressions for a_s and b_s. [9] use 2 in the corresponding place.

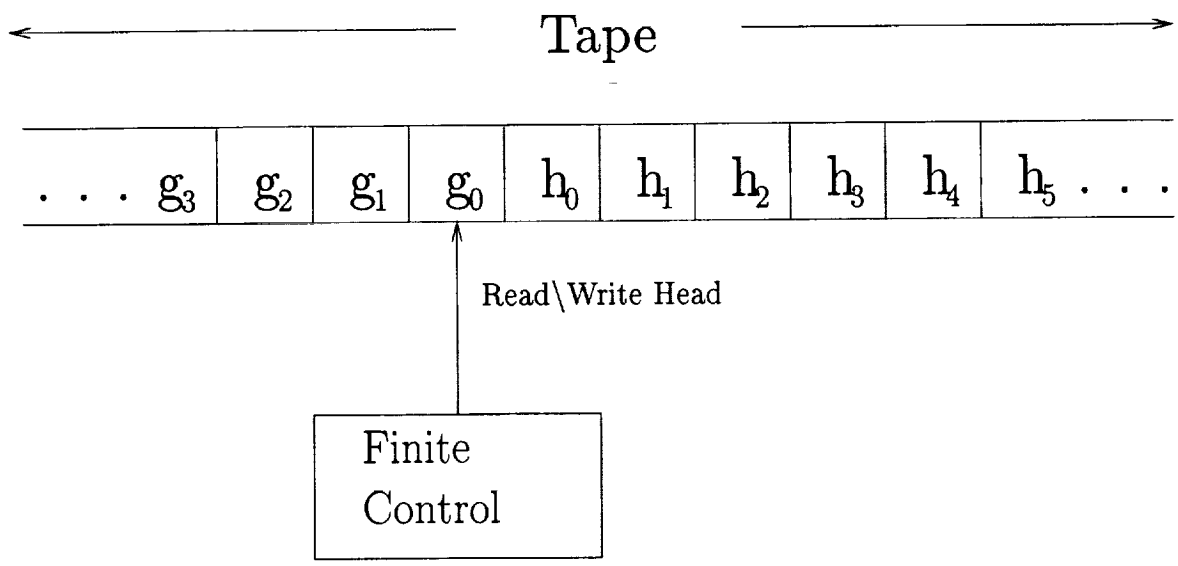

Figure 2: Schematic diagram of a Turing Machine with alphabet $\{0,1\}$.
Note that $g_i, h_i \in \{0,1\}$.

Case 1: $\delta(q,c) = (q', w, L)$
Case 2: $\delta(q,c) = (q', w, R)$

Here, q is the current state of $\mathcal{M}$, q' is the next state of $\mathcal{M}$, $c \in \{0,1\}$ is the symbol below the tape head, and $w \in \{0,1\}$ is the symbol which the tape head writes on the tape. L represents a left move, and R represents a right move. We now map the state transition function δ into the transition operation of an economy as follows:

Case 1: Left move $\delta(q,c) = (q', w, L)$. Let a_b and b_b be the values of a_s and b_s respectively after this transition. Then, a_b and b_b can be written as:

$$a_b = 3\sum_{i=1}^{\infty} \frac{g_i}{3^{i+1}} = 3(a_s - \frac{g_0}{3}) \tag{3}$$

$$b_b = \frac{w}{3} + \frac{1}{3}\sum_{i=0}^{\infty} \frac{h_i}{3^{i+1}} = \frac{w}{3} + \frac{b_s}{3} \tag{4}$$

Case 2: Right move $\delta(q,c) = (q', w, R)$. In this case a_b and b_b can be written as:

$$a_b = \frac{h_0}{3} + \frac{w}{9} + \frac{1}{3}\sum_{i=1}^{\infty} \frac{g_i}{3^{i+1}} = \frac{h_0}{3} + \frac{w}{9} + \frac{a_s - \frac{g_0}{3}}{3} \tag{5}$$

$$b_b = 3\sum_{i=1}^{\infty} \frac{h_i}{3^{i+1}} = 3(b_s - \frac{h_0}{3}) \tag{6}$$

In the next few sections we present agents who perform the very transformations listed above, and then redirect the impulse to the agent corresponding to the new state of $\mathcal{M}$. Thus our economy will simulate the above Turing Machine.

2.3 A Basic Economy

Given the Turing Machine and its encoding, we can construct an economy as follows. Each state of the Turing Machine's finite control is represented by a unique agent of the economy. The computation of the Turing Machine is encoded as an impulse that flows through the economy. At any instant this impulse will reside at exactly one agent. This agent corresponds to the current state of the finite control of the Turing Machine.[5] The magnitude of the impulse represents the value of the tape of the Turing Machine.

When an agent receives an impulse, it optimizes its utility function and transmits a new impulse to one of two[6] neighbors (see figure 3). The choice of the neighbor to whom the impulse is transmitted depends on the state transition function of the Turing Machine as follows. A state transition from σ to σ' of the given Turing Machine is encoded by an impulse transmitted by the agent corresponding to σ towards the agent corresponding to σ'. The optimization function of each agent is designed to ensure that the dynamics of the economy mimic the computation of the given Turing Machine.

Since the number of states of the given Turing Machine is finite, the number of agents in the economy is also finite. The intensity of the received impulse plays an important role in deciding with whom the receiving agent is going to interact. The process of reoptimization may result in the absorption or magnification of some of the intensity of the impulse.[7]

2.4 An Undecidable Economy

We consider the following economy: there are four consumer goods, a, b, c and d, and a leisure good l. The economy consists of a variety of agents who can consume any or all of these goods, but can produce either one unit each of a and b, or one unit each of c and d only. The goods in each production pair are referred to as related goods. Each agent can buy goods from either a producer of a and b, or from a producer of c and d. Goods are bought by transforming leisure into labor, at the constant marginal cost of 1. Figure 3 shows a section of the economy.

[5]This statement has a small technical flaw. In Appendix 3 we discuss this flaw and present a specification for the economy which overcomes this flaw. We have avoided this technical discussion in this section to make this paper easier to read.

[6]The problem can be generalized to a problem of choosing among many choices.

[7]This model is motivated by a discrete-choice making process which results from an impulse. An example is an agent faced with a finite number of possible actions, of which he must choose one. His action causes the agent he interacts with to adjust his behavior, and so on.

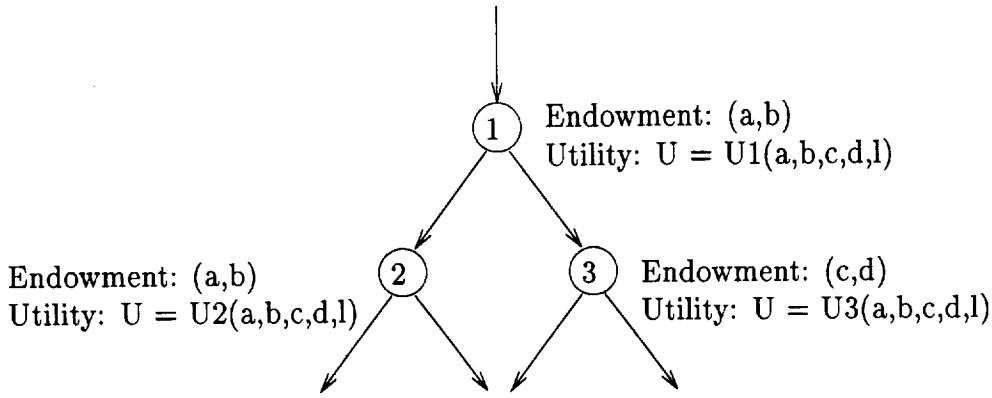

Figure 3: A section of the economy

Following are the variables used in this analysis.

i_s = quantity of good i sold to the preceding agent in a period
i_b = quantity of good i bought from the subsequent agent in a period
i_c = quantity of good i consumed in a period

where i is one of the four consumer goods. The utility function of agent j is of the general form:

$$U_j = \sum_{i=1}^{4} K_{ij} i_c^{\frac{1}{2}} + l \tag{7}$$

where

$$K_{ij} = k(\{i_s\}_{i=1}^{4}). \tag{8}$$

Thus the utility function is continuous and concave in a_c, b_c, c_c and d_c. The parameter K_{ij} is a function of the quantity of produced goods sold by agent j. We motivate this by an 'information effect' on an agent's utility function.

Figure 10 in Appendix 2 shows the different values that K_{ij} can take on for producers of a and b. The corresponding values of K_{ij} for producers of c and d are obtained by replacing a_s with c_s, b_s with d_s, a_b with c_b, etc.

The constant K_{ij} for any good is a function of i_s, where i is the good itself or the related good. The motivation for i_s entering K_{ij} is as follows. An agent's marginal utility for a good depends on how much of the good itself and the related good the agent sold.

We now emphasize some aspects of the economy. Note that all utility functions are concave (weakly so in the case of leisure) and increasing in their arguments. The economy has heterogeneous agents, i.e., different agents may have different utility functions. Thus the economy is consistent with traditional assumptions of economic theory.

We now solve the optimization problem for one possible type of a and b producer. See Appendix 2 for the solutions to the problems for all types of a and b producers. The solution to the problem for the c and d producer is similar and is left to the reader.

The first case of Appendix 2 is solved below. This case corresponds to the following transition in the Turing Machine simulation. The value in the tape cell below the tape head of the Turing Machine is 1. The state transition function of $\mathcal{M}$ for the current state requires the tape head to write a 1 and move to the left.

In this case, the agent chooses to buy a and b. The consumer's problem is:

$$\text{Max } U = 2(2a_s)^{\frac{1}{2}}a_c^{\frac{1}{2}} + 0.1c_c^{\frac{1}{2}} + 2(\frac{4-2b_s}{3})^{\frac{1}{2}}b_c^{\frac{1}{2}} + 0.1d_c^{\frac{1}{2}} + l \tag{9}$$

subject to

$$a_c = 1 - a_s + a_b \tag{10}$$
$$b_c = 1 - b_s + b_b \tag{11}$$
$$c_c = c_b \tag{12}$$
$$d_c = d_b \tag{13}$$

The first order conditions for an optimum are:

$$(2a_s)^{\frac{1}{2}}a_c^{-\frac{1}{2}} = 1 \tag{14}$$

$$(\frac{4-2b_s}{3})^{\frac{1}{2}}b_c^{-\frac{1}{2}} = 1 \tag{15}$$

$$0.05c_c^{-\frac{1}{2}} = 1 \tag{16}$$

$$0.05d_c^{-\frac{1}{2}} = 1 \tag{17}$$

This condition will hold for the two goods which were bought: the agent will not be willing to work for wages beyond this point, because leisure will be more valuable at the margin. Because of the restriction on choice, the first order conditions will not hold for the two goods which were not bought.

If a and b are bought, then $a_c = 2a_s$ and $a_b = 3a_s - 1$. Similarly, $b_c = \frac{2}{3}(2 - b_s)$ and $b_b = \frac{1}{3}(1 + b_s)$. If c and d were to be bought, then $c_c = c_b = \frac{0.1}{2}$ and $d_c = d_b = \frac{0.1}{2}$. It can be checked that the utility from buying c_b and d_b is lower than the utility from buying a_b and b_b. Thus a and b are bought and:

$$a_b = 3a_s - 1 \tag{18}$$

$$b_b = \frac{1 + b_s}{3} \tag{19}$$

Recall that in this case the value of the tape cell below the tape head g_0 is 1. It is easy to verify the the impulse passed on to the next agent, i.e., (a_b, b_b) corresponds to the case in which the tape head writes a 1 and then moves left (see Equations 3 and 4).

The optimization problem for the other types of a and b producer and for the c and d producers can be solved in an analogous manner. A summary of the coefficients k_{ij} for these cases appears in the figure in Appendix 2. Thus we have simulated a Turing Machine using a simplistic economy consisting of producers and consumers who must make discrete choices.

2.5 Correspondence between the Turing Machine and the Economy

In the introduction to this section the stated problem was whether an impulse $I(i)$ reaches an agent α. We can now relate these elements of the economy to the corresponding elements of the given Turing Machine $\mathcal{M}$.

- The agent α corresponds to q_h, the halting state of $\mathcal{M}$.
- The initial shock to the economy is determined as follows. Let i denote the initial value of the tape of $\mathcal{M}$. Then $I(i)$, the initial shock to the economy is (a_s, b_s) as defined in Equations 1 and 2.

In summary, our construction mimics $\mathcal{M}$ as follows. $\mathcal{M}$ halts on input i if and only if an initial shock $I(i)$ reaches α in $\mathcal{E}(x)$.

2.6 The Problem is Recursively Enumerable

In our model economy, the path along which the impulse travels can be partitioned into agent-to-agent subpaths. Since each subpath is an element of the solution to a simple problem, the total path can be determined by a rational agent. Thus the problem is in principle solvable, and a Turing Machine is capable of enumerating all the perturbations which will ultimately affect a given agent. The problem is thus recursively enumerable. In the next section we show that the problem is not recursive, i.e., a Turing Machine is *not* capable of enumerating all the perturbations that *never* affect a given agent.

2.7 Undecidability Proof

We have shown our system and its relation to the Turing Machine. By simulating each state as an agent and routing the impulse between the agents, we can simulate any Turing Machine. Thus we have:

Theorem 1 *The following language is undecidable:*

$$\{\langle I, \mathcal{E}, \alpha\rangle |\ \textit{initial impulse } I \textit{ reaches agent } \alpha \textit{ in economy } \mathcal{E}\}$$

Recall that the Universal Turing Machine can be used to simulate any Turing Machine on any input. Let u denote the specification of any Universal Turing Machine and let

$\mathcal{E}(u)$ denote the economy that mimics u. Let α denote the agent in $\mathcal{E}(u)$ corresponding to the halting states of u. For *any* Turing Machine specification x and input i, ϕ_x halts on input i if and only if the initial shock $I(\langle x, i\rangle)$ reaches α in $\mathcal{E}(u)$. Thus there is a *particular* finite economy $\mathcal{E}(u)$ with a *particular* agent α for which we can show:

Theorem 2 *The following language is undecidable:*

$$\{I|\ \textit{initial impulse } I \textit{ reaches agent } \alpha \textit{ in economy } \mathcal{E}(u)\}$$

We encourage the reader to contrast the statements of Theorems 1 and 2.

3 Applications and Limitations

3.1 Applications

In this section we mention a few areas in which the concept of undecidability is applicable. The areas are trigger problems in macroeconomics, game theory and spectral analysis in econometrics. For all of these examples, consider the kind of network economy presented in section 3.

- **Trigger Problems in a Multi-Sector Economy**[8]
  Consider a node of an economy which releases a significant impulse I_2 if it receives an impulse I_1. If the question of whether impulse I_1 ever passes through the node is undecidable, then no economist or agent with rational expectations will know whether the significant impulse I_2 is ever going to be released. In this framework, our modelling capability and rational expectations are compromised.

- **Game Theory – a dynamic example**
  Consider a repeated dynamic game whose extensive form looks like the economy $\mathcal{E}_T$. Then each node in the tree represents a move by a player. The problem of whether the game passes through a given node is undecidable.

- **Spectral Analysis in Time Series Econometrics**
  In the above economy, the question of whether or not an impulse passes through a certain node of an economy is undecidable. Thus a time series econometrician collecting data at that node can never be sure that (s)he has a complete set of data, because (s)he will never know when to stop collecting data.

The above examples are a small subset of the conceivable range of undecidable problems.

3.2 Limitations

It is important to recognize the limitations of this result. A number of traditional problems are decidable. Examples are certain problems involving homogeneous agents, convergent infinite sequences and finite time horizons. These problems can be solved by a suitably encoded Turing Machine. To the extent that economists are satisfied with the realism of such problems, the issue of undecidability is not important. However, should economists try to model the world as it actually is, in terms of a network of interacting sectors or agents making discrete choices, the issue of undecidability must certainly play a role in their concept of knowledge in economies.

[8]Models which attempt to closely mimic macroeconomic reality, such as Competitive General Equilibrium (CGE) models, are also the types of models which are most susceptible to the problem of undecidability.

4 Conclusion

In this paper we have shown that certain problems in economics are *undecidable*. This provides a formal basis for challenging the validity of the rational expectations hypothesis in the context of such economies. The argument is that there are facts which follow from and only from an agent's information which the agent cannot compute. Philosophically, undecidability illuminates a problem for economists: there are fragments of economics about which it may not be possible to reason.

Appendix 1

In section 2 we showed that the execution of any given Turing Machine can be mimicked by the corresponding economy. In this appendix we illustrate a minor technical flaw in our construction and show how it can be corrected. We first illustrate the flaw with an example.

Suppose q, q_0 and q_1 are states of the given Turing Machine such that $\delta(q,1) = \langle q_1, x, y\rangle$ for some x and y, and $\delta(q,0) = \langle q_0, x', y'\rangle$ for some x' and y'. Then the economy has three agents α, α_1 and α_0 that correspond to the states q, q_1 and q_0 respectively. Our construction requires α_1 to be a producer of a and b, and α_0 to be a producer of c and d.

Further suppose that q' is another state of the given Turing Machine such that $\delta(q',0) = \langle q_1, x, y\rangle$ for some x and y. Our construction requires α_1 to be producer of c and d. In this situation our construction fails. Figure 4 shows the flawed economy.

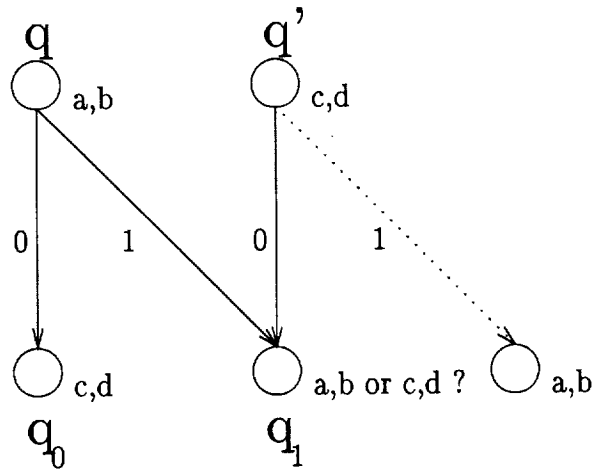

Figure 4: A construction of the economy showing the technical flaw.

We overcome this problem by converting the Turing Machine

$$\mathcal{T} = \langle Q, \Gamma, \perp, \Sigma, \delta, q_0, q_a, q_r\rangle$$

into the modified Turing Machine

$$\mathcal{T}' = \langle Q', \Gamma, \perp, \Sigma, \delta', \langle q_0, 0\rangle, \langle q_a, 0\rangle, \langle q_r, 0\rangle\rangle$$

as follows:

- $Q' = Q \times \{0,1\}$. In other words, every state q of $\mathcal{M}$ is represented by *two* states of $\mathcal{M}'$: $\langle q, 0\rangle$ and $\langle q, 1\rangle$.

- δ' mimics δ as follows. Let q be any state of $\mathcal{M}$. Suppose that $\mathcal{M}$ changes state from q to q' upon reading i, i.e., $\delta(q,i) = \langle q', x, y\rangle$ for some x and y. Then $\delta'(\langle q, 0\rangle, i) = \delta'(\langle q, 1\rangle, i) = \langle\langle q', i\rangle, x, y\rangle$.

- To correctly address the halting condition, we also add the following transitions:

$$\delta'(\langle q_a, 1\rangle, 0) = \delta'(\langle q_a, 1\rangle, 1) = \langle\langle q_a, 0\rangle, 1, R\rangle$$

$$\delta'(\langle q_r, 1\rangle, 0) = \delta'(\langle q_r, 1\rangle, 1) = \langle\langle q_r, 0\rangle, 1, R\rangle$$

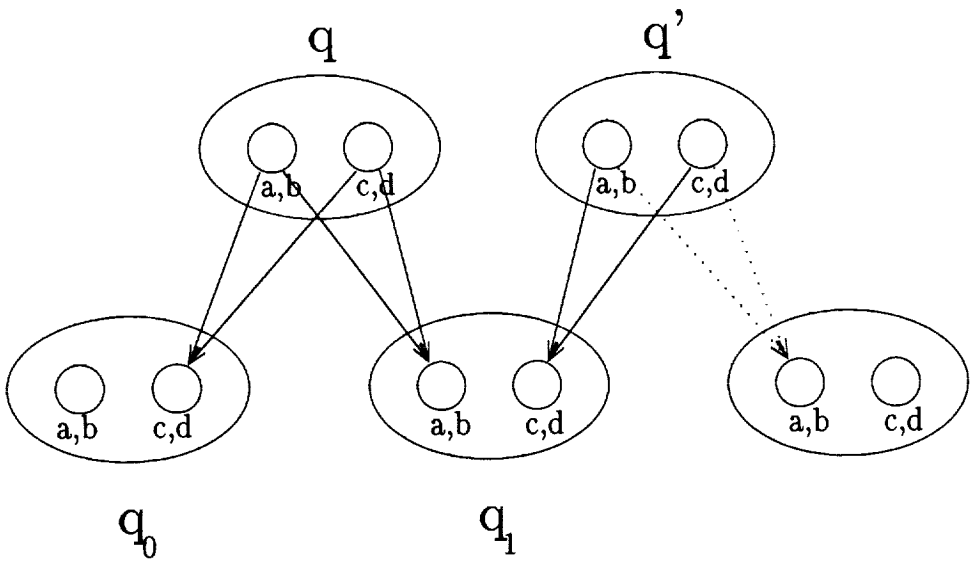

Figure 5: A construction of the economy used to overcome the technical flaw.

Figure 5 is a visual representation of the above construction. It is easy to verify that $L(\mathcal{M}') = L(\mathcal{M})$ and that our construction works correctly with $\mathcal{M}'$. In summary, given any arbitrary Turing Machine $\mathcal{M}$, we first modify it into another Turing Machine $\mathcal{M}'$ as described in this appendix and then convert $\mathcal{M}'$ into an economy using the techniques of section 2.

Appendix 2

Tape Head Action			Corresponding Action of Agent j							
Read	Write	Move	Condition	Choice	a_b	b_b	$K_{aj} = a_c$	$K_{bj} = b_c$	$K_{cj} = c_c$	$K_{dj} = d_c$
1	1	L	$a_s \geq \frac{1}{3}$	a, b	$3a_s - 1$	$\frac{1+b_s}{3}$	$2a_s$	$\frac{4-2b_s}{3}$	0	0
1	0	L	$a_s \geq \frac{1}{3}$	a, b	$3a_s - 1$	$\frac{b_s}{3}$	$2a_s$	$\frac{3-2b_s}{3}$	0	0
1	1	R	$a_s \geq \frac{1}{3}$	a, b	$\frac{a_s+\lfloor 3b_s \rfloor}{3}$ †	$\frac{9b_s-\lfloor 3b_s \rfloor}{3}$	$\frac{3-2a_s+\lfloor 3b_s \rfloor}{3}$	$1 + 2b_s - \lfloor 3b_s \rfloor$	0	0
1	0	R	$a_s \geq \frac{1}{3}$	a, b	$\frac{a_s+\lfloor 3b_s \rfloor-\frac{1}{3}}{3}$	$\frac{9b_s-\lfloor 3b_s \rfloor}{3}$	$\frac{8-6a_s+\lfloor 3b_s \rfloor}{3}$	$1 + 2b_s - \lfloor 3b_s \rfloor$	0	0
0	1	L	$a_s < \frac{1}{3}$	c, d	0	0	$1 - a_s$	$1 - b_s$	$3a_s$	$\frac{1+b_s}{3}$
0	0	L	$a_s < \frac{1}{3}$	c, d	0	0	$1 - a_s$	$1 - b_s$	$3a_s$	$\frac{b_s}{3}$
0	1	R	$a_s < \frac{1}{3}$	c, d	0	0	$1 - a_s$	$1 - b_s$	$\frac{1+a_s+\lfloor 3b_s \rfloor}{3}$	$\frac{9b_s-\lfloor 3b_s \rfloor}{3}$
0	0	R	$a_s < \frac{1}{3}$	c, d	0	0	$1 - a_s$	$1 - b_s$	$\frac{a_s+\lfloor 3b_s \rfloor}{3}$	$\frac{9b_s-\lfloor 3b_s \rfloor}{3}$

Figure 10: Possible parameters for utility functions of producers of a and b.

† $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x. Note that $\lfloor 3b_s \rfloor = h_0$ of Fig 6.

Appendix 3

Optimization for the Recursion Theorist[9]

An Economy

An economy consists of a set of people or groups of people, referred to as *agents*, that produce, trade and consume *goods*. The *endowment* of an agent at any time is the set of goods produced or acquired by that agent prior to that time and currently within its possession. For example, the endowment of a farmer today may be 100 tons of rice, the endowment of an investor may be twenty shares in Imperial Chemical Industries etc.

Each agent in the economy consumes goods to satisfy its needs or to derive pleasure. By consuming different mixes of goods, an agent derives different amounts of *utility*. For example, consuming 1 apple and 1 orange might yield a different utility from consuming 2 apples and 1 banana. The utility of an agent is quantitatively captured by a *utility function*. Each agent tries to *maximize* its utility function. Note that (1) different agents may have different utility functions and (2) the utility function of any given agent may be affected by the past behavior of that agent and by the behavior of other agents.

Utility Maximization

In practice, economists assume that utility functions are concave.[10] This captures the observation that in most cases, every additional unit of a good consumed gives less additional pleasure.

In order to maximize their utility function, agents may trade a part of their endowment with each other. For example, consider an economy in which there are two agents A and O. A is an apple farmer whose endowment is 10 apples and O is an orange farmer whose endowment is 10 oranges. A's utility function U_A is

$$U_A(a, o) = 2a^{\frac{1}{2}} + o^{\frac{1}{2}}$$

and O's utility function U_O is

$$U_O(a, o) = 2a^{\frac{1}{2}} + 3o^{\frac{1}{2}}$$

If A and O do not trade, A's utility will be $U_A(10, 0) = 2\sqrt{10} \approx 6.3$. Similarly, O's utility will be approximately 9.5. Suppose A and O agree to trade 1 apple for 1 orange, A's utility will be $U_A(9, 1) = 7$ and O's utility will be $U_O(9, 1) = 11$—both benefit from the transaction. In fact, A would be even happier if (s)he could trade 2 apples for 2 oranges.[11] In order to determine when A will be happiest, we need to solve the following constrained optimization problem for a and o:

[9] [11] provides a thorough treatment of this subject.

[10] This convenient assumption has never been proved, and is not always validated by observed behavior.

[11] To keep things simple in this example, we assume that 1 apple is always traded for exactly 1 orange.

$$\text{Max} U_A(a, o) = 2a^{\frac{1}{2}} + o^{\frac{1}{2}}$$

subject to

$$a + o = 10,$$

known as the budget constraint and

$$a \geq 0, o \geq 0$$

known as nonnegativity constraints.

Using the method of Lagrange, we derive

$$\frac{\partial U_A(a, o)}{\partial a} = \frac{\partial U_A(a, o)}{\partial o},$$

known as the marginality condition.

The function $\frac{\partial U_A(a,o)}{\partial a}$ is referred to as the apple farmer's *marginal utility of a* and $\frac{\partial U_A(a,o)}{\partial o}$ is referred to as the apple farmer's *marginal utility of o*. In most cases,[12] given a concave utility function f in n goods $g_1, g_2, \ldots g_n$, the maximum utility can be determined by equating all n marginal utilities of f (i.e., with respect to each of the n goods).

In this paper we consider a *finite* economy, i.e., one with a finite number of agents. Each agent has a set of *neighbors* with whom it can perform transactions. This is indeed the case in practice, since an agent can only interact with other agents that are "nearby". In other words, if the physical or temporal distance between two agents is sufficiently large, they cannot trade with each other.

We assume that time progresses in discrete units. In each time unit, an agent makes a decision based on its endowment and its utility function, and may participate in a transaction with one of its neighbors.

We subject this economy to an exogenous shock. An example of an exogenous shock is an increase in the endowment of an agent. In practice, this may be achieved by reduced taxation, low-interest loans, an outright grant etc. As economists, we wish to determine what effect that this action will have on the economy. In particular, we wish to determine whether an agent A will be affected if an agent B's endowment is increased. In practice, this ability helps in policy decision making. For example, consider a government that would like ensure that all its citizens receive at least a subsistence amount of food. One possible way of achieving this goal would be to provide a subsidy to all farmers in the economy and hope that some of the benefit in reduced food production "costs" is passed on to the poor.

Thus the government wants to solve the following problem: if the endowment of the farming sector is increased, will the poor benefit sufficiently? More generally, we can study the following abstract problem: if the endowment of an agent A is increased, will another agent B be affected?

[12] A common exception is a "corner" solution.

We show that in general, this problem is *undecidable.* Given an arbitrary Turing Machine $\mathcal{T}$, we show how to construct an economy $\mathcal{E}_{\mathcal{T}}$ in which an agent B is affected by the perturbation if and only if $\mathcal{T}$ halts on an empty tape. Technically, this result is similar to showing that a ray tracing problem is undecidable [9].

Appendix 4

Introducing Recursion Theory to Economists

Introduction

In this section, we go over some key concepts in recursion theory. An understanding of these concepts is necessary for an understanding of the result of this paper. First, we define some terms necessary for an understanding of recursion theory. Then, as a step toward understanding the Turing Machine, we present a weaker and simpler computational device called the Deterministic Finite Automaton. The concept of a Deterministic Finite Automaton also serves to emphasize the power of the Turing Machine. Finally, we introduce the Turing Machine and state some fundamental theorems that we will use to derive our result.

This section summarizes the exposition on Deterministic Finite Automata and Turing Machines in [5, 6]: we make extensive use of the language and organization of [5, 6].

Basic Definitions

In this section we introduce the economist to Recursion Theory. We provide the definitions of terms used in sections 4 and 4. A more detailed treatment of these concepts is available in [5].

Definition 1 *An* alphabet *is a finite set of symbols.*

For example consider the set of equations in elementary calculus. The alphabet for this set, denoted Σ_i, contains the following symbols:

1. The digits, $0, 1, \ldots 9$.
2. The decimal point, ".".
3. A *finite* set of connectives and operators, $+, -, \times, \div, \hat{}, =, (,), \partial, d, \int, \lim, \rightarrow$, sin, cos, etc.
4. A *finite* set of variables x, y, etc.
5. A *finite* set of constants π, e, etc.

Clearly, this set of symbols is finite. We use the symbol Σ to denote an arbitrary alphabet.

Definition 2 *A* string *over an alphabet Σ is a finite list of symbols belonging to the set Σ.*

For example,

$$dy \div dx = x\hat{}2 + e\hat{}(-x) \tag{20}$$

is a string over Σ_i, the alphabet for the set of equations in elementary calculus.

Definition 3 *The* concatenation *of two strings x and y is denoted by xy.*

Definition 4 *Σ^* denotes the set of all strings over the alphabet Σ.*

Definition 5 *The* length *of a string x, denoted $|x|$ is the number of symbols in x.*

For example, the length of the string in Equation 20 is 16.

Clearly not all strings over Σ_i belong to the set of true equations in elementary calculus. For example consider the following strings:

1. "$x($" is not well-formed. It violates the rule that the number of open parentheses in an equation should equal the number of closed parentheses in an equation.

2. "$1 = 2$" while well-formed, is false.

Informally, the concept of a language is used to differentiate between "acceptable" and "unacceptable" elements of Σ^*. Thus the two unacceptable strings above do not belong to the language of equations in elementary calculus.

Definition 6 *A* language *over Σ is a subset of Σ^*.*

A decision problem is specified by a set A of all possible inputs and $B \subseteq A$ of "acceptable" instances. For example the following is a decision problem: is a specific sector in an economy affected by a trade shock. The set of inputs is the set of all possible trade shocks to the economy. The subset of "acceptable" instances is the set of all trade shocks which affect the specified sector.

Definition 7 *A* decision problem *over Σ is a function from Σ^* that returns a "yes" or "no" answer.*

Systems, States and Transitions

Intuitively, a system is a set of agents and relations connecting the agents. The *state* of a system is a complete, instantaneous description of the system. In particular, the state of a system provides all relevant information about each agent in the system and the relations connecting them at the given instant.

Every system, is governed by a set of laws called the *state transition function* of that system. The state transition function determines how the state of the system evolves over time. For simplicity, we assume that the state transitions occur instantaneously.

As an example, we consider the following economy with n agents that trade in a common market. Each agent a has a utility function U_a which it tries to maximize. The endowment of a at time t is denoted by $E_a(t)$. At the end of time period t, agent a selects x_a units of one commodity c_a from its endowment and exchanges it for other commodities from other agents. In exchange for c_a, agent a acquires a basket of goods, b_a.

The state of this system at time t is an n-tuple $\langle E_{a_1}(t), E_{a_2}(t), ..., E_{a_n}(t)\rangle$. The state transition function of this system is the solution of the utility maximization problem for each agent. Thus if we are given each agent's utility function and the current state of this system, then we can determine how the system will evolve over time.

The Deterministic Finite Automaton

In this section we briefly describe a simple class of computing devices known as Deterministic Finite Automata. We use some of the notation of [5] and [6].

Definition 8 *A* Deterministic Finite Automaton *is a 5-tuple*

$$\mathcal{M} = \langle Q, \Sigma, \delta, q_0, A \rangle$$

where:

1. Q is the finite set of states of $\mathcal{M}$.
2. Σ is the finite alphabet of $\mathcal{M}$.
3. $\delta : Q \times \Sigma \rightarrow Q$ is the state transition function of $\mathcal{M}$.
4. $q_0 \in Q$ is the initial state of $\mathcal{M}$.
5. $A \subseteq Q$ is the set of accepting states of $\mathcal{M}$.

The Deterministic Finite Automaton $\mathcal{M}$ works as follows. It starts in its initial state, q_0. At each time period, it reads one symbol from its input and changes its state based on its state transition function. We say that $\mathcal{M}$ *accepts* if at the end of its input it is in an accepting state, i.e., a state in A. Otherwise, we say that $\mathcal{M}$ *rejects*. Thus the set of inputs on which $\mathcal{M}$ accepts defines a language on Σ.

Definition 9 *If on input x, $\mathcal{M}$ accepts, we say $\mathcal{M}$* accepts *x.*

Definition 10 *The language of a Deterministic Finite Automaton $\mathcal{M}$, denoted $L(\mathcal{M})$, is $\{x \in \Sigma^* | \mathcal{M}$ accepts $x\}$.*

Definition 11 *A set $S \subseteq \Sigma^*$ is called* regular *if and only if there is a Deterministic Finite Automaton $\mathcal{M}$ such that $L(\mathcal{M}) = S$.*

An Example

As an example consider $\mathcal{M}$, a Deterministic Finite Automaton that accepts $\{x \in \{a,b\}^* | x$ does not contain the substring $aba\}$. The set of states of $\mathcal{M}$ is $\{q_0, q_1, q_2, q_3\}$. The alphabet of $\mathcal{M}$ is $\{a, b\}$. The initial state of $\mathcal{M}$ is q_0. The set of accepting states of $\mathcal{M}$ is $\{q_0, q_1, q_2\}$. The state transition function of $\mathcal{M}$ is given in figure 6. To illustrate how $\mathcal{M}$ functions, we show all the state transitions of $\mathcal{M}$ when its input is *aaabba*. See figure 7. When $\mathcal{M}$ reads the last symbol of the input (i.e., a), $\mathcal{M}$ enters the state q_1. $\mathcal{M}$ accepts *aaabba* because q_1 is an accepting state of $\mathcal{M}$.

As mentioned earlier, Deterministic Finite Automata have limited computing power. This is illustrated by a theorem known as the pumping lemma. This theorem states that there is no Deterministic Finite Automaton that accepts certain sets of strings that are very simple to specify. We next explain the pumping lemma briefly and show that the set $\{x | x = a^n b^n$ for some $n\}$ is not regular.

State	Input	
	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_3	q_0
q_3	q_3	q_3

Figure 6: State transition function for Deterministic Finite Automaton $\mathcal{M}$

State	Input Symbol	Next State
q_0	a	q_1
q_1	a	q_1
q_1	a	q_1
q_1	b	q_2
q_2	b	q_0
q_0	a	q_1

Figure 7: Deterministic Finite Automaton $\mathcal{M}$ responding to *aaabba*

The Pumping Lemma

The intuition for the pumping lemma is the following. Consider a Deterministic Finite Automaton $\mathcal{M}$ such that $L(\mathcal{M})$ is infinite. Since $\mathcal{M}$ consists of a finite set of states, there exists a sufficiently long string $x \in L(\mathcal{M})$ that will force the automaton to repeat at least one state. Thus x is of the form abc where q, the state of $\mathcal{M}$ after scanning a, is the state of $\mathcal{M}$ after scanning ab. Since $\mathcal{M}$ cannot tell the difference between two visitations of the same state, the state of $\mathcal{M}$ after scanning abb must be q. Thus the substring between the two visitations of the repeated state can be inserted into the string before the first occurrence of the state and the newly constructed string will still be accepted.

This observation can be used to show that the set $S = \{x | x \text{ is of the form } a^n b^n\}$[13] is not regular, i.e., there is no Deterministic Finite Automaton that accepts precisely this set. This is proven by contradiction: suppose there exists a Deterministic Finite Automaton $\mathcal{M}$ such that $L(\mathcal{M}) = S$. Take a string of the form $a^m b^m$ such that m is greater than the number of states in $\mathcal{M}$. There must exist x and $y, x < y < m$ such that the state of $\mathcal{M}$ after it scans a^x, is the same as the state of $\mathcal{M}$ after it scans a^y. We leave it to the reader to verify that $\mathcal{M}$ must accept $a^{m+y-x}b^m$, a string that is clearly not in S.

[13] a^n denotes a string of n consecutive a's. a^* denotes a string of any number of consecutive a's.

Turing Machines, Computability and Undecidability

Introduction

The Turing Machine was invented in 1936 by Alan Turing [14]. It was conceived at a time when mathematicians were trying to define the concept of computability of functions. Examples of such efforts are Church's λ-calculus [2], Post's Post systems [8], Godel's μ-recursive functions [3], etc. Later mathematicians showed that all of these systems are computationally equivalent, leading Church to declare that each system captured the *intuitive* notion of "computable". This declaration is known as Church's Thesis. Church's Thesis is widely accepted by Recursion Theorists, and there is no known computable function that cannot be programmed by a Turing Machine. Recursion Theorists use the Turing Machine to define computability. Informally, a language L is said to be computable if and only if there is a Turing Machine that accepts L.

An Informal Description of Turing Machines

In this section, we provide an informal description of Turing Machines. Essentially, a Turing Machine is a Deterministic Finite Automaton (see page 21) augmented with a one-way infinite 1 - dimensional tape on which it may read and write values (see figure 8). The tape of the Turing Machine is divided into an infinite number of *tape cells*, each of which contains a symbol in Γ, an alphabet that contains Σ and a blank symbol $\perp \notin \Sigma$. The Turing Machine accesses the tape via a single *tape head.* The Turing Machine may read, write or overwrite a symbol on the tape cell beneath the tape head. A state transition for a Turing Machine consists of a change in the state of the Deterministic Finite Automaton associated with the Turing Machine, a command to write a symbol in the cell below the tape head, and a command to move the tape head one cell to the left or right. The input to the Turing Machine is a finite string from Σ^* and is initially written on the tape in contiguous tape cells (see figure 8). The infinitely many cells on either side of the input are assigned the blank symbol $\perp$.

The Turing Machine starts in its initial state q_0 with its head scanning the leftmost symbol of the input. In each step the Turing Machine reads the symbol on the tape cell beneath its tape head, and depending on that symbol and the current state of the Turing Machine, it writes a new symbol on that tape cell, moves its tape head either one cell left or right and enters some new state. The action taken by the Turing Machine in each situation is determined by its state transition function δ. It *accepts* its input by entering a special accept state q_a and *rejects* its input by entering a special reject state q_r. The Turing Machine is said to *halt* on input x if it either accepts x or rejects x. Note that it may do neither, by running infinitely on input x without ever accepting or rejecting.

A Formal Description of Turing Machines

Definition 12 *A Turing Machine [5] is a 8-tuple*

$$\mathcal{M} = \langle Q, \Gamma, \perp, \Sigma, \delta, q_0, q_a, q_r \rangle$$

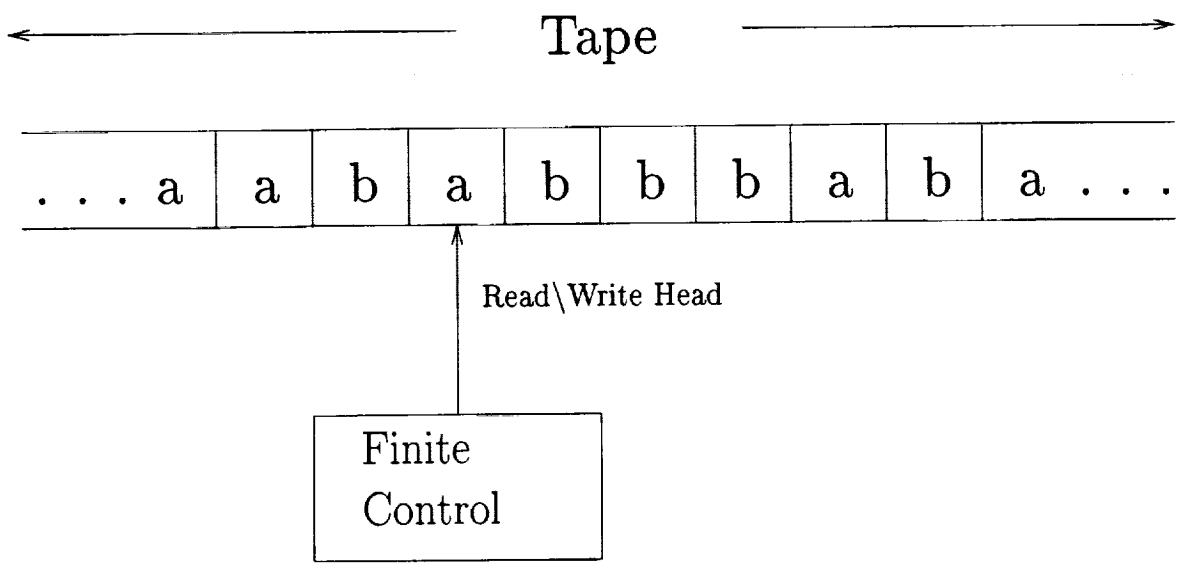

Figure 8: Schematic diagram of a Turing Machine with $\Sigma = \{a, b\}$.

where:

1. Q is the finite set of states of $\mathcal{M}$.
2. Γ is the set of allowable tape symbols of $\mathcal{M}$.
3. $\perp$ is the blank symbol of $\mathcal{M}$.
4. Σ is the set of input symbols of $\mathcal{M}$. Note that $\Sigma \subseteq \Gamma - \{\perp\}$.[14]
5. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the state transition function of $\mathcal{M}$.
6. $q_0 \in Q$ is the initial state of $\mathcal{M}$.
7. $q_a \in Q$ is the accepting state of $\mathcal{M}$.
8. $q_r \in Q$ is the rejecting state of $\mathcal{M}$.

Figure 8 is a schematic diagram of a Turing Machine. Q and q_0 for a Turing Machine are the same as Q and q_0 for the Deterministic Finite Automaton of the Turing Machine. The input to the Turing Machine is in Σ^* for some alphabet Σ. Each tape cell contains one symbol from a set Γ, a superset of Σ that contains $\perp$, the special blank symbol. The state transition function of the Turing Machine now returns a triple rather than just the

[14] In some cases, Σ is a proper subset of $\Gamma - \{\perp\}$. In our example in figure 9, $\Gamma = \Sigma \cup \{\perp, \clubsuit\}$.

state of the Deterministic Finite Automaton of the Turing Machine. As before, $\delta(q, a)$ describes the actions of the Turing Machine when it is in state q and the tape cell below its tape head contains the value a. If $\delta(q, a) = \langle q', a', d \rangle$, then the operation of the Turing Machine is as follows.

1. The Turing Machine writes a' on the tape cell below its head.
2. The Turing Machine moves the tape head in direction d (either left, denoted L or right, denoted R).
3. The Turing Machine enters state q'.

In addition, once the machine enters the accept q_a or the reject state q_r, its execution *halts*. If it halts in q_a, we say that it *accepts*. If it halts in q_r, we say that it *rejects*.

Definition 13 *If on input x, $\mathcal{M}$ accepts, we say $\mathcal{M}$* accepts x.

Definition 14 *If on input x, $\mathcal{M}$ rejects, we say $\mathcal{M}$* rejects x.

Definition 15 *The language of a Turing Machine $\mathcal{M}$, denoted $L(\mathcal{M})$, is $\{x \in \Sigma^* | \mathcal{M}$ accepts $x\}$.*

An Example

As an example, consider $\mathcal{M}$, a Turing Machine that accepts the set $\{a^n b^n c^n | n \geq 0\}$. Σ, the set of input symbols of $\mathcal{M}$ is $\{a, b, c\}$. The blank symbol of $\mathcal{M}$ is $\perp$. Apart from $\perp$ and the symbols in Σ, Γ contains the symbol $\clubsuit$ called the erased symbol. Thus the set of allowable tape symbols of $\mathcal{M}$ is $\{a, b, c, \perp, \clubsuit\}$.

Informally, $\mathcal{M}$ repeats the following procedure as often as possible:

1. If there does not exist at least one each of a, b and c, then $\mathcal{M}$ stops and decides.
2. If there exists at least one each of a, b and c, the head goes from left to right over the input, erasing the first a, the first b and the first c.
3. The head goes back to the beginning of the input.

At the end, if there is any symbol in $\{a, b, c\}$ left, then clearly the string is not of the form $a^n b^n c^n$. Also note that care is taken to avoid strings of the form *abaccb*, i.e., strings with an equal number of a's, b's and c's but not of the form $a^* b^* c^*$. The state transition function of $\mathcal{M}$ is given in figure 9.

Note that the above set is not regular. That is, we can use the pumping lemma to show that there is no Deterministic Finite Automaton that will accept this set. The reader is encouraged to verify that $\mathcal{M}$ actually accepts $a^n b^n c^n$ by trying out a few examples.

State	a	b	c	$\perp$	$\clubsuit$
q_0	$\langle q_1, \clubsuit, R\rangle$	$\langle q_6, b, R\rangle$	$\langle q_6, c, R\rangle$	$\langle q_5, \perp, R\rangle$	$\langle q_0, \clubsuit, R\rangle$
q_1	$\langle q_1, a, R\rangle$	$\langle q_2, \clubsuit, R\rangle$	$\langle q_6, c, R\rangle$	$\langle q_6, \perp, R\rangle$	$\langle q_1, \clubsuit, R\rangle$
q_2	$\langle q_6, a, R\rangle$	$\langle q_2, b, R\rangle$	$\langle q_3, \clubsuit, R\rangle$	$\langle q_6, \perp, R\rangle$	$\langle q_2, \clubsuit, R\rangle$
q_3	$\langle q_6, a, R\rangle$	$\langle q_6, b, R\rangle$	$\langle q_3, c, R\rangle$	$\langle q_4, \perp, L\rangle$	$\langle q_3, \clubsuit, R\rangle$
q_4	$\langle q_4, a, L\rangle$	$\langle q_4, b, L\rangle$	$\langle q_4, c, L\rangle$	$\langle q_0, \perp, R\rangle$	$\langle q_4, \clubsuit, L\rangle$
q_5	This is the accept state				
q_6	This is the reject state				

Figure 9: State transition function for Turing Machine $\mathcal{M}$

Decision Problems and Recursive Functions

In the previous sections, we saw Turing Machines that check whether a given string is in a language. A Turing Machine can also be used to compute a function as follows. The tape of the Turing Machine is initialized to contain the parameters of the function; the tape head initially scans the leftmost symbol of the input. The output of the function is taken to be the value written on the tape when the Turing Machine goes into its accepting state.

Definition 16 *A function f is* recursive *if and only if there is a Turing Machine $\mathcal{M}$ such that on every input x in the domain of f, $\mathcal{M}$ eventually writes $f(x)$ on its tape and enters its accepting state.*

Given a Turing Machine $\mathcal{M}$ that computes a function f and a Turing Machine $\mathcal{M}'$ that solves a decision problem, we can "concatenate" $\mathcal{M}$ and $\mathcal{M}'$ as follows. We can construct a Turing Machine $\mathcal{M}^+$ that first runs $\mathcal{M}$. When $\mathcal{M}$ enters its accepting state, $\mathcal{M}^+$ runs $\mathcal{M}'$. Thus $\mathcal{M}^+$ accepts $\{x | f(x) \in L(\mathcal{M}')\}$. More precisely, given

$$\mathcal{M} = \langle Q, \Gamma, \perp, \Sigma, \delta, q_0, q_a, q_r\rangle$$

and

$$\mathcal{M}' = \langle Q', \Gamma, \perp, \Sigma, \delta', q_0', q_a', q_r'\rangle$$

Assuming for simplicity that Q and Q' are disjoint, we can construct

$$\mathcal{M}^+ = \langle Q^+, \Gamma, \perp, \Sigma, \delta^+, q_0^+, q_a^+, q_r^+\rangle$$

where

1. $Q^+ = Q \cup Q' - \{q_a, q_r\}$
2. $\delta^+(q,a) = \begin{cases} \delta'(q,a) & \text{if } q \in Q' \\ \langle q', a', d\rangle & \text{if } \delta(q,a) = \langle q', a', d\rangle \text{ and } q' \neq q_a \\ \langle q_0', a', d\rangle & \text{if } \delta(q,a) = \langle q', a', d\rangle \text{ and } q' = q_a \end{cases}$

3. $q_0^+ = q_0$

4. $q_a^+ = q_a'$ and $q_r^+ = q_r'$.

$\mathcal{M} \bullet \mathcal{M}'$ denotes the concatenation of $\mathcal{M}$ and $\mathcal{M}'$.

Decidability and Undecidability

Definition 17 *A set $S \subseteq \Sigma^*$ is called* recursively enumerable *(abbreviated r.e.) if and only if there is a Turing Machine $\mathcal{M}$ such that $L(\mathcal{M}) = S$.*

Definition 18 *A set $S \subseteq \Sigma^*$ is called* recursive *if both S and $\overline{S}$ are recursively enumerable .*[15]

A property is *decidable* if there exists a Turing Machine that accepts all strings with that property and rejects all strings without that property.

Definition 19 *A property is decidable if and only if the set of all elements having that property is recursive.*

A property is *undecidable* if there is no Turing Machine that can determine whether an arbitrary given string has that property.

Certain properties are undecidable. For example, there is no mechanical procedure that can be used to tell whether or not a string causes a Turing Machine to halt. In particular, there is no finite set of axioms and inference rules that can be used to determine which strings and Turing Machines have this property and which do not.

Note that the terms "recursive" and "recursively enumerable" apply to sets. The terms "decidable" and "undecidable" apply to properties of elements of sets. By a slight abuse of terminology, we say that a *language is undecidable* if there is no Turing Machine that can determine whether an arbitrary given string belongs to that language. [4] contains a good intuitive description of the above concepts.

Universal Turing Machines

As we saw earlier, a Turing Machine is specified by an 8-tuple $\langle Q, \Gamma, \perp, \Sigma, \delta, q_0, q_a, q_r \rangle$. It is easy to encode this specification so that the only symbols that occur in it are "0" and "1". Thus the specification of the Turing Machine can be given as a string over the alphabet $\{0, 1\}$.

With this encoding technique in mind, we can view the set of specifications of all Turing Machines as a language over $\{0, 1\}$. That is, each specification of a Turing Machine corresponds to a unique binary number. The Turing Machine specification corresponding to the number x is denoted ϕ_x.

Note that ϕ_x is not defined for all x since some numbers do not correspond to a Turing Machine specification. This is notationally inconvenient. Thus if x does not correspond

[15] $\overline{S}$ is used to denote the complement of the set S.

to a Turing Machine specification, ϕ_x is defined (by default) to be the Turing Machine that accepts $\emptyset$.

A *Universal Turing Machine* is a Turing Machine whose language is the set of pairs of Turing Machines $\mathcal{M}$ and strings x such that $x \in L(\mathcal{M})$. Intuitively, the Universal Turing Machine is capable of simulating *every* Turing Machine. More precisely, the alphabet of a Universal Turing Machine is $\{0,1\}$. The Universal Turing Machine accepts the string $\langle x, y \rangle$ if and only if ϕ_x accepts y. To see the construction of a Universal Turing Machine, the reader is referred to [5].

Since the Universal Turing Machine is capable of simulating every Turing Machine, it is capable of simulating itself. The property by which a Turing Machine can simulate itself is called *self reference*. This allows us to use a proof technique called *diagonalization* to prove that some problems are undecidable.

Diagonalization

We first illustrate the method of diagonalization with what is perhaps its simplest example.

Theorem 3 *There is an English sentence that is neither true nor false.*

PROOF: Consider the sentence:

"This sentence is false."

Let us assume for contradiction that this sentence is either true or false. There are two cases to consider:

1. The sentence is true. In this case, the sentence is false—a contradiction.
2. The sentence is false. In this case, it is false that the sentence is false, i.e., the sentence is true—a contradiction. □

Note that the above sentence refers to itself.

The technique of diagonalization was first used by Cantor [4] at the end of the last century to show that there does not exist a one-to-one correspondence between the natural numbers N and 2^N, the power set of N, which is defined as follows.

Definition 20 *The* power set *of a set A, denoted 2^A, is the set of all the subsets of A.*

Cantor's argument is as follows: Suppose for a contradiction that there is a one-to-one and onto function

$$f : N \rightarrow 2^N.$$

	1	2	3	4	5	6	7	8	9	...
$f(1)$	1	0	0	1	0	0	1	0	0	...
$f(2)$	1	0	0	1	1	1	0	0	1	...
$f(3)$	0	0	0	0	1	1	0	0	1	...
$f(4)$	0	0	0	0	1	1	1	0	0	...
$f(5)$	1	1	1	1	1	1	1	0	0	...
$f(6)$	0	1	1	1	1	0	1	0	1	...
$f(7)$	0	1	0	1	0	1	0	1	0	...
$f(8)$	1	0	1	0	1	0	1	0	1	...
$f(9)$	1	0	1	0	0	1	1	0	1	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Figure 10: An example of M showing the relationship between N and 2^N.

We can build the following infinite two-dimensional matrix, M:

$$M_{i,j} = \begin{cases} 1 & \text{if } j \in f(i) \\ 0 & \text{if } j \notin f(i) \end{cases}$$

In other words, each row of M represents a different function. The 1st row denotes $f(1)$, the second row denotes $f(2)$, and so on. For example, figure 10 represents one possible matrix M. $f(1) = \{1, 4, 7, \ldots\}$, $f(2) = \{1, 4, 5, 6, 9, \ldots\}$, and so on. By our assumption that f is onto, every subset of N appears as a row of the matrix.

Now we will use diagonalization to derive a contradiction. Construct the complement of the infinite string down the main diagonal of the matrix by switching the 1's in it to 0's and the 0's to 1's. In other words, construct $S \subset N$ as follows:

$$i \in S \text{ if and only if } M_{i,i} = 0 \text{ (i.e., } i \notin f(i))$$

The set S for the matrix M in figure 10 is $\{2, 3, 4, 6, 7, 8, \ldots\}$. Since f is one-to-one onto and $S \subset N$, there must be a number i such that $f(i) = S$. There are two cases to consider:

1. $i \in f(i)$. In this case, $M_{i,i} = 1$. Hence $i \notin S$. Since $f(i) = S$, we have a contradiction.

2. $i \notin f(i)$. In this case, $M_{i,i} = 0$. Hence $i \in S$. Since $f(i) = S$, we have a contradiction.

Note that this argument can applied to a wide variety of sets other than N. A similar argument was also used in "Russell's paradox" [4].

Undecidability of the Halting Problem

Cantor's simple cardinality argument allows us to prove the existence of undecidable decision problems. Informally, the proof is as follows: there are uncountably many languages over $\{0,1\}^*$ but only countably many Turing Machines. Thus there must exist many languages that are not accepted by any Turing Machine.

In this section, we prove the undecidability of one such problem, known as the *Halting Problem* and denoted H [5]. Informally, H is stated as follows: given a Turing Machine specification i and a string x, does ϕ_i halt on x? More formally H is the language defined as follows:

$$H = \{\langle i, x\rangle | \phi_i(x) \text{ halts}\}$$

Theorem 4 *The halting problem is undecidable.*

PROOF: Suppose for a contradiction that there is a Turing Machine specification h such that ϕ_h accepts $\langle i, x\rangle$ if ϕ_i halts on x and ϕ_h rejects $\langle i, x\rangle$ otherwise. ϕ_h can be represented as the following infinite two-dimensional matrix:

$$\phi_h(i,j) \begin{cases} \textit{accepts} \text{ if } \phi_i \text{ halts on } j \\ \textit{rejects} \text{ if } \phi_i \text{ does not halt on } j \end{cases}$$

Now we will use diagonalization to derive a contradiction. Consider the following function *dbl* defined as follows:

$$dbl(x) = \langle x, x\rangle$$

dbl is a recursive function. Let $h' = \langle Q, \Gamma, \perp, \Sigma, \delta, q_0, q_a, q_r\rangle$ represent the specification of the Turing Machine $dbl \bullet \phi_h$. Then $\phi_{h'}$ behaves as follows:

$$\phi_{h'}(i) \begin{cases} \textit{accepts} \text{ if } \phi_i \text{ halts on } i \\ \textit{rejects} \text{ if } \phi_i \text{ does not halt on } i \end{cases}$$

Note that $\phi_{h'}$ corresponds to the diagonal of the two-dimensional matrix representing ϕ_h. We can construct a Turing Machine specification h^+ based on the specification h' as follows. Add a new state q_n and replace all transitions to q_a with transitions to q_n. Thus $\phi_{h'}$ and ϕ_{h^+} behave identically except when $\phi_{h'}$ enters q_a, ϕ_{h^+} enters q_n. More formally, we construct $h^+ = \langle Q^+, \Gamma, \perp, \Sigma, \delta^+, q_0, q_a, q_r\rangle$ as follows:

1. $Q^+ = Q \cup q_n$, where q_n is a new state not in Q
2. $\delta^+(q,a) = \begin{cases} \langle q', a', d\rangle & \text{if } \delta(q,a) = \langle q', a', d\rangle, q' \neq q_a \text{ and } q \neq q_n \\ \langle q_n, a', d\rangle & \text{if } \delta(q,a) = \langle q_a, a', d\rangle \text{ and } q \neq q_n \\ \langle q_n, a, R\rangle & \text{if } q = q_n \end{cases}$

It is easy to verify the following facts:

- h^+ is the specification of a Turing Machine that does not accept any strings. In other words, $L(\phi_{h^+}) = \emptyset$.
- If $\phi_{h'}$ rejects string x, then ϕ_{h^+} also rejects x.
- If $\phi_{h'}$ accepts string x, then ϕ_{h^+} does not halt on x. In fact ϕ_{h^+} enters state q_n and never leaves that state.

Intuitively ϕ_{h^+} halts on x if and only if ϕ_x does not halt on x. Let us now consider what happens when we run ϕ_{h^+} with h^+ as input. There are two cases to consider:

1. ϕ_{h^+} halts on h^+. In this case, ϕ_{h^+} does not halt on h^+—a contradiction.
2. ϕ_{h^+} does not halt on h^+. In this case, ϕ_{h^+} halts on h^+—a contradiction.

We conclude from the above argument that H is not recursive. □

In other words, there does not exist a Turing Machine that can determine in a finite amount of time whether any given Turing Machine $\mathcal{M}$ halts on an arbitrary given input x.

Reducibility: A Tool to Prove Undecidability

In this section we introduce *reducibility*, a relation that allows us to compare degrees of difficulty of solving problems. Using reducibility, it is possible to show that many problems are at least as hard to solve as the Halting Problem. Since we know that the Halting Problem is undecidable, it follows that all these problems are also undecidable.

We now define reducibility.

Definition 21 *Let L_1 and L_2 be any two languages over some finite alphabet Σ. We say that L_1 is* Turing reducible *to L_2, written $L_1 <_T L_2$ if and only if there is a recursive function $F : \Sigma^* \rightarrow \Sigma^*$ that maps elements of L_1 to elements of L_2 and elements of $\overline{L_1}$ to elements of $\overline{L_2}$.*

In other words, there is a Turing Machine $\mathcal{M}$ with the following properties. If the input to $\mathcal{M}$ is an element of L_1, $\mathcal{M}$ writes an element of L_2 on its tape before halting. If the input to $\mathcal{M}$ is an element of $\overline{L_1}$, $\mathcal{M}$ writes an element of $\overline{L_2}$ on its tape before halting. Note that several types of reducibility have been studied in the literature. For this paper, however, we will only be concerned with Turing reducibility. For the rest of this paper, we use the term "reducibility" to denote Turing reducibility. We can prove the following theorem about the halting problem H:

Theorem 5 *For any language $\mathcal{L}$, if $H <_T \mathcal{L}$, then $\mathcal{L}$ is not recursive.*

PROOF: Suppose for a contradiction that $\mathcal{L}$ is recursive; there must be a Turing Machine $\mathcal{M}$ that halts on every input, such that $L(\mathcal{M}) = \mathcal{L}$. Further, since H is reducible to $\mathcal{L}$, there is a Turing Machine $\mathcal{M}'$ with the following properties. If the input to $\mathcal{M}'$ is an

element of H, $\mathcal{M}'$ eventually writes an element of $\mathcal{L}$ on its tape and halts. If the input to $\mathcal{M}'$ is an element of $\overline{H}$, $\mathcal{M}'$ eventually writes an element of $\overline{\mathcal{L}}$ on its tape and halts.

Consider the Turing Machine $\mathcal{M}^+ = \mathcal{M}' \bullet \mathcal{M}$. We will show that $\mathcal{M}^+$ halts on every input and $L(\mathcal{M}^+) = H$. Since the halting problem is undecidable, this immediately gives us a contradiction. There are two cases to consider:

1. The input to $\mathcal{M}^+$ is an element of H. In this case, $\mathcal{M}^+$ works as follows. First $\mathcal{M}'$ runs until completion. Since the input to $\mathcal{M}'$ is an element of H, $\mathcal{M}'$ writes x, an element of $\mathcal{L}$ on its tape and halts. Next $\mathcal{M}$ runs until completion. Since x, the input to $\mathcal{M}$, is an element of $\mathcal{L}$ and $L(\mathcal{M}) = \mathcal{L}$, $\mathcal{M}$ accepts. Thus $\mathcal{M}^+$ accepts.

2. The input to $\mathcal{M}^+$ is an element of $\overline{H}$. In this case, $\mathcal{M}^+$ works as follows. First $\mathcal{M}'$ runs until completion. Since the input to $\mathcal{M}'$ is an element of $\overline{H}$, $\mathcal{M}'$ writes x, an element of $\overline{\mathcal{L}}$ on its tape and halts. Next $\mathcal{M}$ runs until completion. Since x, the input to $\mathcal{M}$, is an element of $\overline{\mathcal{L}}$ and $\mathcal{M}$ halts on every input, $\mathcal{M}$ rejects. Thus $\mathcal{M}^+$ rejects.

We have shown that $\mathcal{M}'$ halts on every input and that $L(\mathcal{M}') = H$, giving the desired contradiction. □

In section 2 we will define a problem $\mathcal{P}$ in a finite economy and show that the Halting Problem is reducible to $\mathcal{P}$. From Theorem 5, it follows that $\mathcal{P}$ is undecidable.

On the Robustness of the Turing Machine model

Even though the Turing Machine model appears to be simple, it is very powerful. In this subsection we mention a few of the enhancements of the Turing Machine model that do not increase its power. We hope that this gives the reader some intuition for the robustness of the Turing Machine model:

Two-way infinite tapes: The Turing Machine model presented in this paper assumes that the tape is infinite in only one direction. With this enhancement, we allow the tape to be infinite in both directions.

Multiple tapes: With this enhancement, the Turing Machine is allowed to have several tapes, each of which has its own tape head. Each tape can be controlled independently.

Multiple tape heads: With this enhancement, the Turing Machine is allowed to have many tape heads on each tape. This allows the Turing Machine to read different cells of the tape simultaneously. Each tape head can be controlled independently.

Multi-dimensional tapes: The Turing Machine model presented in this paper assumes that the tape is 1-dimensional. With this enhancement, we allow the tape to be multi-dimensional and infinite in all dimensions.

By introducing non-determinism: The Turing Machine model presented in this paper is completely *deterministic*, i.e., based on the current state of a Turing Machine and its state transition function, it is possible to determine the next state of the Turing Machine. There are several ways to bypass this restriction. One obvious way is to allow the Turing Machine to toss a coin at each step and make its transition based on (1) its current state, (2) the value in the tape cell below its tape head and (3) the result of the coin toss. Other ways of eliminating determinism such as *non-determinism* and *alternation* have also been considered. For a detailed description, the reader is referred to [5].

None of the enhancements mentioned above increase the power of the Turing Machine model. Furthermore, even if we apply a combination of these enhancements, the computing power of the Turing Machine model remains unchanged. More precisely, given any Turing Machine $\mathcal{M}$ that uses some or all of the enhancements mentioned above, there exists a Turing Machine $\mathcal{M}'$ without any enhancements such that (a) $\mathcal{M}'$ accepts if and only if $\mathcal{M}$ accepts, (b) $\mathcal{M}'$ rejects if and only if $\mathcal{M}$ rejects and (c) $\mathcal{M}'$ neither accepts nor rejects if and only if $\mathcal{M}$ neither accepts nor rejects. The proof of this fact is beyond the scope of this paper: for this, the reader is referred to [5].

Acknowledgments

We would like to thank Karl Shell for his valuable advice and comments on the result. We would like to thank Mike Reiter, Sonal Deshpande and Nish Shah for their useful comments on previous drafts of this paper and James Grosjean for his thorough reading and revealing comments. Siddharth thanks John Curran, Mark Fisher and James Grosjean for expanding the bounds of his rationality.

References

[1] Ken Binmore. Debayesing game theory, June 1991. Lecture for the International Conference on Game Theory, Florence.

[2] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.

[3] Kurt Gödel. *On Formally Undecidable Propositions.* Basic Books, New York, NY, 1962.

[4] Douglas R. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid.* Random House, Inc., New York, NY, 1989.

[5] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, 1979.

[6] Dexter Kozen. Lecture notes for introduction to theory of computation. Lecture notes for Cornell University course CS381, 1992.

[7] Robert E. Lucas Jr. On the mechanics of economic development. *Journal of Monetary Economics*, 22, 1988.

[8] E. Post. Finite combinatory processes — formulation, I. *Journal of Symbolic Logic*, 1, 1936.

[9] John H. Reif, J.D. Tygar, and Akitoshi Yoshida. The computability and complexity of optical beam tracing. In *Proceedings of the Thirty-first Symposium on Foundations of Computer Science*, pages 106–114, June 1990.

[10] Thomas J. Sargent. *Dynamic Macroeconomic Theory.* Harvard University Press, Cambridge MA, 1987.

[11] Eugene Silberberg. *The structure of economics : a mathematical analysis.* McGraw-Hill, 1978.

[12] Stephen E. Spear. Learning rational expectations under computability constraints. *Econometrica*, 57(4):889–910, 1989.

[13] Robert Townsend. Models of money with spatially separated agents. In J.H. Kareken and Neil Wallace, editors, *Models of Monetary Economies*, pages 265–304. Federal Reserve Bank of Minneapolis, Minneapolis, 1980.

[14] A.M. Turing. On computable numbers with an application to the Entscheidungsproblem. *Proceedings of the London Math. Society*, 2, 1936.

[15] Neil Wallace. *Models of Overlapping Generations: An Exposition.* University of Minnesota, Minneapolis MN, 1978.

[16] Henry Y. Wan (Jr.). The new classical economics—a game-theoretic critique. In George R Feiwel, editor, *Issues in contemporary macroeconomics and distribution*, page 237. State University of New York Press, Albany, 1985.