Algorithmic Rationality: Game Theory with Costly Computation

Joseph Y. Halpern
Cornell University
halpern@cs.cornell.edu

Rafael Pass
Cornell University
rafael@cs.cornell.edu

First Draft: April 2007
This Draft: December 2009*

Abstract

We develop a general game-theoretic framework for reasoning about strategic agents performing possibly costly computation. In this framework, many traditional game-theoretic results (such as the existence of a Nash equilibrium) no longer hold. Nevertheless, we can use the framework to provide psychologically appealing explanations of observed behavior in well-studied games (such as finitely repeated prisoner's dilemma and rock-paper-scissors). Furthermore, we provide natural conditions on games sufficient to guarantee that equilibria exist.

*Minor updates in September 2011. An extended abstract of this paper appeared in the 1st Innovations in Computer Science Conference, 2010. Halpern is supported in part by NSF grants ITR-0325453, IIS-0534064, IIS-0812045, and IIS-0911036, and by AFOSR grants FA9550-08-1-0438 and FA9550-09-1-0266, and ARO grant W911NF-09-1-0281. Pass is supported in part by a Microsoft Research Faculty Fellowship, NSF CAREER Award CCF-0746990, AFOSR Award FA9550-08-1-0197, and BSF Grant 2006317.

1 Introduction

Consider the following game. You are given a random odd n-bit number x and you are supposed to decide whether x is prime or composite. If you guess correctly you receive \$2, if you guess incorrectly you instead have to pay a penalty of \$1000. Additionally you have the choice of "playing safe" by giving up, in which case you receive \$1. In traditional game theory, computation is considered "costless"; in other words, players are allowed to perform an unbounded amount of computation without it affecting their utility. Traditional game theory suggests that you should compute whether x is prime or composite and output the correct answer; this is the only Nash equilibrium of the one-person game, no matter what n (the size of the prime) is. Although for small n this seems reasonable, when n grows larger most people would probably decide to "play safe"; eventually the cost of computing the answer (e.g., by buying powerful enough computers) outweighs the possible gain of \$1.

The importance of considering such computational issues in game theory has been recognized since at least the work of Simon [1955]. There have been a number of attempts to capture various aspects of computation. Two major lines of research can be identified. The first line, initiated by Neyman [1985], tries to model the fact that players can do only bounded computation, typically by modeling players as finite automata. Neyman focused on finitely repeated prisoner's dilemma, a topic which has continued to attract attention. (See [Papadimitriou and Yannakakis 1994] and the references therein for more recent work on prisoner's dilemma played by finite automata; Megiddo and Wigderson [1986] considered prisoner's dilemma played by Turing machines.) In another instance of this line of research, Dodis, Halevi, and Rabin [2000] and Urbano and Vila [2004] consider the problem of implementing mediators when players are polynomial-time Turing machines.

The second line, initiated by Rubinstein [1986], tries to capture the fact that doing costly computation affects an agent's utility. Rubinstein assumed that players choose a finite automaton to play the game rather than choosing a strategy directly; a player's utility depends both on the move made by the automaton and the complexity of the automaton (identified with the number of states of the automaton). Intuitively, automata that use more states are seen as representing more complicated procedures. (See [Kalai 1990] for an overview of the work in this area in the 1980s, and [Ben-Sasson, Kalai, and Kalai 2007] for more recent work.)

Our focus is on providing a general game-theoretic framework for reasoning about agents performing costly computation. As in Rubinstein's work, we view players as choosing a machine, but for us the machine is a Turing machine, rather than a finite automaton. We associate a complexity, not just with a machine, but with the machine and its input. The complexity could represent the running time of or space used by the machine on that input. The complexity can also be used to capture the complexity of the machine itself (e.g., the number of states, as in Rubinstein's case) or to model the cost of searching for a new strategy to replace one that the player already has. For example, if a mechanism designer recommends that player i use a particular strategy (machine) M, then there is a cost for searching for a better strategy; switching to another strategy may also entail a psychological cost. By allowing the complexity to depend on the machine *and* the input, we can deal with the fact that machines run much longer on some inputs than on others. A player's utility depends both on the actions chosen by all the players' machines and the complexity of these machines.

In this setting, we can define Nash equilibrium in the obvious way. However, as we show by a simple example (a rock-paper-scissors game where randomization is costly), a Nash equilibrium may not always exist. Other standard results in game theory, such as the *revelation principle* (which, roughly speaking, says that there is always an equilibrium where players truthfully report their types, i.e., their private information [Myerson 1979; Forges 1986]) also do not hold. We view this as a feature. We believe that taking computation into account should force us to rethink a

number of basic notions. To this end, we introduce refinements of Nash equilibrium that take into account the computational aspects of games. Moreover, as we show here by example, despite the fact that Nash equilibrium may not always exist, there are interesting Nash equilibria that give insight into a number of games.

First, we show that the non-existence of Nash equilibrium is not such a significant problem. We identify natural conditions (such as the assumption that randomizing is free) that guarantee the existence of Nash equilibrium in computational games. Moreover, we demonstrate that our computational framework can explain experimentally-observed phenomena, such as cooperation in the finitely repeated prisoner's dilemma, that are inconsistent with standard Nash equilibrium in a psychologically appealing way. Indeed, as shown by our results, many concerns expressed by the emerging field of *behavioral economics* (pioneered by Kahneman and Tversky [1981]) can be accounted for by simple assumptions about players' cost of computation, without resorting to ad hoc cognitive or psychological models.

Finally, we show that our framework is normative in that it can be used to tell a mechanism designer how to design a mechanism that takes into account agents' computational limitations. We illustrate this point in the context of cryptographic protocols. In a companion paper [Halpern and Pass 2009a], we use our framework to provide a game-theoretic definition of cryptographic protocol security.

Paper Outline The rest of this paper is organized as follows. In Sections 2 and 3 we describe our framework. In Section 4 show how our computational framework can provide "psychologically-appealing" explanations to observed behavior that is inconsistent with traditional solution concepts. Section 5 contains our existence results for Nash equilibrium. We conclude in Section 7 with new directions of research made possible by our framework.

2 A Computational Game-Theoretic Framework

2.1 Bayesian Games

We model costly computation using *Bayesian machine games*. To explain our approach, we first review the standard notion of a *Bayesian game*. A Bayesian game is a game of incomplete information, where each player makes a single move. The "incomplete information" is captured by assuming that nature makes an initial move, and chooses for each player i a *type* in some set T_i. Player i's type can be viewed as describing i's private information. For ease of exposition, we assume in this paper that the set N of players is always $[m] = \{1, \ldots, m\}$, for some m. If $N = [m]$, the set $T = T_1 \times \ldots \times T_m$ is the *type space*. As is standard, we assume that there is a commonly-known probability distribution Pr on the type space T. Each player i must choose an action from a space A_i of actions. Let $A = A_1 \times \ldots \times A_m$ be the set of action profiles. A Bayesian game is characterized by the tuple $([m], T, A, \Pr, \vec{u})$, where $[m]$ is the set of players, T is the type space, A is the set of joint actions, and $\vec{u}$ is the utility function, where $u_i(\vec{t}, \vec{a})$ is player i's utility (or payoff) if the type profile is $\vec{t}$ and action profile $\vec{a}$ is played.

In general, a player's choice of action will depend on his type. A *strategy* for player i is a function from T_i to $\Delta(A_i)$ (where, as usual, we denote by $\Delta(X)$ the set of distributions on the set X). If σ is a strategy for player i, $t \in T_i$, and $a \in A_i$, then $\sigma(t)(a)$ denotes the probability of action a according to the distribution on acts induced by $\sigma(t)$. Given a joint strategy $\vec{\sigma}$, we can take $u_i^{\vec{\sigma}}$ to be the random variable on the type space T defined by taking $u_i^{\vec{\sigma}}(\vec{t}) = \sum_{\vec{a} \in A}(\sigma_1(t_1)(a_1) \times \ldots \times \sigma_m(t_m)(a_m))u_i(\vec{t}, \vec{a})$. Player i's expected utility if $\vec{\sigma}$ is played, denoted $U_i(\vec{\sigma})$, is then just $\mathbf{E}_{\Pr}[u_i^{\vec{\sigma}}] = \sum_{\vec{t} \in T} \Pr(\vec{t}) u_i^{\vec{\sigma}}(\vec{t})$.

2.2 Bayesian Machine Games

In a Bayesian game, it is implicitly assumed that computing a strategy—that is, computing what move to make given a type—is free. We want to take the cost of computation into account here. To this end, we consider what we call *Bayesian machine games*, where we replace strategies by *machines*. For definiteness, we take the machines to be Turing machines, although the exact choice of computing formalism is not significant for our purposes. Given a type, a strategy in a Bayesian game returns a distribution over actions. Similarly, given as input a type, the machine returns a distribution over actions. Just as we talk about the expected utility of a strategy profile in a Bayesian game, we can talk about the expected utility of a machine profile in a Bayesian machine game. However, we can no longer compute the expected utility by just taking the expectation over the action profiles that result from playing the game. A player's utility depends not only on the type profile and action profile played by the machine, but also on the "complexity" of the machine given an input. The complexity of a machine can represent, for example, the running time or space usage of the machine on that input, the size of the program description, or some combination of these factors. For simplicity, we describe the complexity by a single number, although, since a number of factors may be relevant, it may be more appropriate to represent it by a tuple of numbers in some cases. (We can, of course, always encode the tuple as a single number, but in that case, "higher" complexity is not necessarily worse.) Note that when determining player i's utility, we consider the complexity of all machines in the profile, not just that of i's machine. For example, i might be happy as long as his machine takes fewer steps than j's.

We assume that nature has a type in $\{0,1\}^*$. While there is no need to include a type for nature in standard Bayesian games (we can effectively incorporate nature's type into the type of the players), once we take computation into account, we obtain a more expressive class of games by allowing nature to have a type (since the complexity of a machine on a given input may depend on nature's type). We assume that machines take as input strings of 0s and 1s and output strings of 0s and 1s. Thus, we assume that both types and actions can be represented as elements of $\{0,1\}^*$. We allow machines to randomize, so given a type as input, we actually get a distribution over strings. To capture this, we assume that the input to a machine is not only a type, but also a string chosen with uniform probability from $\{0,1\}^\infty$ (which we can view as the outcome of an infinite sequence of coin tosses). The machine's output is then a deterministic function of its type and the infinite random string. We use the convention that the output of a machine that does not terminate is a fixed special symbol ω. We define a *view* to be a pair (t,r) of two bitstrings; we think of t as that part of the type that is read, and of r as the string of random bits used. (Our definition is slightly different from the traditional way of defining a view, in that we include only the parts of the type and the random sequence *actually* read. If computation is not taken into account, there is no loss in generality in including the full type and the full random sequence, and this is what has traditionally been done in the literature. However, when computation is costly, this may no longer be the case.) We denote by $t;r$ a string in $\{0,1\}^*;\{0,1\}^*$ representing the view. (Note that here and elsewhere, we use ";" as a special symbol that acts as a separator between strings in $\{0,1\}^*$.) If $v = (t;r)$ and r is a finite string, we take $M(v)$ to be the output of M given input type t and random string $r \cdot 0^\infty$.

We use a *complexity function* $\mathscr{C} : \mathbf{M} \times \{0,1\}^*; (\{0,1\}^* \cup \{0,1\}^\infty) \to I\!N$, where $\mathbf{M}$ denotes the set of Turing machines, to describe the complexity of a machine given a view. If $t \in \{0,1\}^*$ and $r \in \{0,1\}^\infty$, we identify $\mathscr{C}(M,t;r)$ with $\mathscr{C}(M,t;r')$, where r' is the finite prefix of r actually used by M when running on input t with random string r.

For now, we assume that machines run in isolation, so the output and complexity of a machine does not depend on the machine profile. We remove this restriction in the next section, where we allow machines to communicate with mediators (and thus, implicitly, with each other via the

mediator).

Definition 2.1 (Bayesian machine game) *A* Bayesian machine game G *is described by a tuple* $([m], \mathcal{M}, T, \Pr, \mathscr{C}_1, \ldots, \mathscr{C}_m, u_1, \ldots, u_m)$*, where*

- $[m] = \{1, \ldots, m\}$ *is the set of players;*
- $\mathcal{M}$ *is the set of possible machines;*
- $T \subseteq (\{0,1\}^*)^{m+1}$ *is the set of type profiles, where the* $(m+1)$*st element in the profile corresponds to nature's type;*
- $\Pr$ *is a distribution on* T*;*
- $\mathscr{C}_i$ *is a complexity function;*
- $u_i : T \times (\{0,1\}^*)^m \times \mathbb{N}^m \to \mathbb{R}$ *is player* i*'s utility function. Intuitively,* $u_i(\vec{t}, \vec{a}, \vec{c})$ *is the utility of player* i *if* $\vec{t}$ *is the type profile,* $\vec{a}$ *is the action profile (where we identify* i*'s action with* M_i*'s output), and* $\vec{c}$ *is the profile of machine complexities.*

We can now define the expected utility of a machine profile. Given a Bayesian machine game $G = ([m], \mathcal{M}, \Pr, T, \vec{\mathscr{C}}, \vec{u})$ and $\vec{M} \in \mathcal{M}^m$, define the random variable $u_i^{G,\vec{M}}$ on $T \times (\{0,1\}^\infty)^m$ (i.e., the space of type profiles and sequences of random strings) by taking

$$u_i^{G,\vec{M}}(\vec{t}, \vec{r}) = u_i(\vec{t}, M_1(t_1; r_1), \ldots, M_m(t_m; r_m), \mathscr{C}_1(M_1, t_1; r_1), \ldots, \mathscr{C}_m(M_m, t_m)).$$

Note that there are two sources of uncertainty in computing the expected utility: the type t and realization of the random coin tosses of the players, which is an element of $(\{0,1\}^\infty)^k$. Let $\Pr_U^k$ denote the uniform distribution on $(\{0,1\}^\infty)^k$. Given an arbitrary distribution $\Pr_X$ on a space X, we write $\Pr_X^{+k}$ to denote the distribution $\Pr_X \times \Pr_U^k$ on $X \times (\{0,1\}^\infty)^k$. If k is clear from context or not relevant, we often omit it, writing $\Pr_U$ and $\Pr_X^+$. Thus, given the probability $\Pr$ on T, the expected utility of player i in game G if $\vec{M}$ is used is the expectation of the random variable $u_i^{G,\vec{M}}$ with respect to the distribution $\Pr^+$ (technically, $\Pr^{+m}$): $U_i^G(\vec{M}) = \mathbf{E}_{\Pr^+}[u_i^{G,\vec{M}}]$. This notion of utility allows an agent to prefer a machine that runs faster to one that runs slower, even if they give the same output, or to prefer a machine that has faster running time to one that gives a better output. Because we allow the utility to depend on the whole profile of complexities, we can capture a situation where i can be "happy" as long as his machine runs faster than j's machine. Of course, an important special case is where i's utility depends only on his own complexity. All of our results continue to hold if we make this restriction.

2.3 Nash Equilibrium in Machine Games

Given the definition of utility above, we can now define (ϵ-) Nash equilibrium in the standard way.

Definition 2.2 (Nash equilibrium in machine games) *Given a Bayesian machine game* $G = ([m], \mathcal{M}, T, \Pr, \vec{\mathscr{C}}, \vec{u})$*, a machine profile* $\vec{M} \in \mathcal{M}^m$*, and* $\epsilon \geq 0$*,* M_i *is an* ϵ*-best response to* $\vec{M}_{-i}$ *if, for every* $M_i' \in \mathcal{M}$*,*

$$U_i^G[(M_i, \vec{M}_{-i})] \geq U_i^G[(M_i', \vec{M}_{-i})] - \epsilon.$$

(As usual, $\vec{M}_{-i}$ *denotes the tuple consisting of all machines in* $\vec{M}$ *other than* M_i*.)* $\vec{M}$ *is an* ϵ-Nash equilibrium *of* G *if, for all players* i*,* M_i *is an* ϵ*-best response to* $\vec{M}_{-i}$*. A* Nash equilibrium *is a 0-Nash equilibrium.*

There is an important conceptual point that must be stressed with regard to this definition. Because we are implicitly assuming, as is standard in the game theory literature, that the game

is common knowledge, we are assume that the agents understand the costs associated with each Turing machine. That is, they do not have to do any "exploration" to compute the costs. In addition, we do not charge the players for computing which machine is the best one to use in a given setting; we assume that this too is known. This model is appropriate in settings where the players have enough experience to understand the behavior of all the machines on all relevant inputs, either through experimentation or theoretical analysis. We can easily extend the model to incorporate uncertainty, by allowing the complexity function to depend on the state (type) of nature as well as the machine and the input. For example, if we take complexity to be a function of running time, a machine M has a true running time for a given input, but a player may be uncertain as to what it is. We can model this uncertainty by allowing the complexity function to depend on the state of nature. To simplify the exposition, we do not have the complexity function depend on the state of nature in this paper; we do so in [Halpern and Pass 2010], where the uncertainty plays a more central role. However, we remark that it is straightforward to extend all the results in this paper to the setting where there is uncertainty about the complexity and the "quality" of a TM.

One immediate advantage of taking computation into account is that we can formalize the intuition that ϵ-Nash equilibria are reasonable, because players will not bother changing strategies for a gain of ϵ. Intuitively, the complexity function can "charge" ϵ for switching strategies. Specifically, an ϵ-Nash equilibrium $\vec{M}$ can be converted to a Nash equilibrium by modifying player i's complexity function to incorporate the overhead of switching from M_i to some other strategy, and having player i's utility function decrease by $\epsilon' > \epsilon$ if the switching cost is nonzero; we omit the formal details here. Thus, the framework lets us incorporate explicitly the reasons that players might be willing to play an ϵ-Nash equilibrium. This justification of ϵ-Nash equilibrium seems particularly appealing when designing mechanisms (e.g., cryptographic protocols) where the equilibrium strategy is made "freely" available to the players (e.g., it is accessible on a web-page), but any other strategy requires some implementation cost.

Although the notion of Nash equilibrium in Bayesian machine games is defined in the same way as Nash equilibrium in standard Bayesian games, the introduction of complexity leads to some significant differences in their properties. We highlight a few of them here. First, note that our definition of a Nash equilibrium considers *behavioral strategies*, which in particular might be randomized. It is somewhat more common in the game-theory literature to consider *mixed strategies*, which are probability distributions over deterministic strategies. As long as agents have perfect recall, mixed strategies and behavioral strategies are essentially equivalent [Kuhn 1953]. However, in our setting, since we might want to charge for the randomness used in a computation, such an equivalence does not necessarily hold.

Mixed strategies are needed to show that Nash equilibrium always exists in standard Bayesian games. As the following example shows, since we can charge for randomization, a Nash equilibrium may not exist in a Bayesian machine game, even in games where the type space and the output space are finite.

Example 2.3 (Rock-paper-scissors) Consider the 2-player Bayesian game of roshambo (rock-paper-scissors). Here the type space has size 1 (the players have no private information). We model playing rock, paper, and scissors as playing 0, 1, and 2, respectively. The payoff to player 1 of the outcome (i, j) is 1 if $i = j \oplus 1$ (where $\oplus$ denotes addition mod 3), -1 if $j = i \oplus 1$, and 0 if $i = j$. Player 2's playoffs are the negative of those of player 1; the game is a zero-sum game. As is well known, the unique Nash equilibrium of this game has the players randomizing uniformly between 0, 1, and 2.

Now consider a machine game version of roshambo. Suppose that we take the complexity of a deterministic strategy to be 1, and the complexity of a strategy that uses randomization to be 2, and take player i's utility to be his payoff in the underlying Bayesian game minus the complexity of

his strategy. Intuitively, programs involving randomization are more complicated than those that do not randomize. With this utility function, it is easy to see that there is no Nash equilibrium. For suppose that (M_1, M_2) is an equilibrium. If M_1 uses randomization, then 1 can do better by playing the deterministic strategy $j \oplus 1$, where j is the action that gets the highest probability according to M_2 (or is the deterministic choice of player 2 if M_2 does not use randomization). Similarly, M_2 cannot use randomization. But it is well known (and easy to check) that there is no equilibrium for roshambo with deterministic strategies. (Of course, there is nothing special about the costs of 1 and 2 for deterministic vs. randomized strategies. This argument works as long as all three deterministic strategies have the same cost, and it is less than that of a randomized strategy.)

The non-existence of Nash euqilibrium does not depend on charging directly for randomization; it suffices that it be difficult to compute the best randomized response. Consider the variant where we do not charge for the first, say, 10,000 steps of computation, and after that there is a positive cost of computation. It is not hard to show that, in finite time, using coin tossing with equal likelihood of heads and tails, we cannot exactly compute a uniform distribution over the three choices, although we can approximate it closely.[1] For example, if we toss a coin twice, playing rock if we get heads twice, paper with heads and tails, and scissors with tails and heads, and try again with two tails, we do get a uniform distribution, conditional on termination. Since there is always a deterministic best reply that can be computed quickly ("play rock", "play scissors", or "play paper"), from this observation it easily follows, as above, that there is no Nash equilibrium in this game either. As a corollary, it follows that that there are computational games without a Nash equilibrium where all constant-time strategies are taken to be free. It is well known that people have difficulty simulating randomization; we can think of the cost for randomizing as capturing this difficulty. Interestingly, there are roshambo tournaments (indeed, even a Rock Paper Scissors World Championship), and books written on roshambo strategies [Walker and Walker 2004]. Championship players are clearly not randomizing uniformly (they could not hope to get a higher payoff than an opponent by randomizing). Our framework provides a psychologically plausible account of this lack of randomization.

The key point here is that, in standard Bayesian games, to guarantee equilibrium requires using randomization. Here, we allow randomization, but we charge for it. This charge may prevent there from being an equilibrium. ∎

Example 2.3 shows that sometimes there is no Nash equilibrium. It is also trivial to show that given any standard Bayesian game G (without computational costs) and a *computable* strategy profile $\vec{\sigma}$ in G (where $\vec{\sigma}$ is computable if, for each player i and type t of player i, there exists a Turing machine M that outputs a with probability $\sigma_i(t)(a)$), we can choose computational costs and modify the utility function in G in such a way as to make $\vec{\sigma}$ a dominant-strategy equilibrium of the modified game: we simply make the cost to player i of implementing a strategy other than σ_i sufficiently high.

One might be tempted to conclude from these examples that Bayesian machine games are uninteresting. They are not useful prescriptively, since they do not always provide a prescription as to the right thing to do. Nor are they useful descriptively, since we can always "explain" the use of a particular strategy in our framework as the result of a high cost of switching to another strategy. With regard to the first point, as we show in a companion paper [Halpern and Pass 2009a], our framework can provide useful prescriptive insights in the context of cryptographic protocols. Moreover, although there may not always be a Nash equilibrium, it is easy to see that there is always an ϵ-Nash equilibrium for some ϵ; this ϵ-Nash can give useful guidance into how the game

[1] Consider a probabilistic Turing machine M with running time bounded by T that outputs either 0, 1, or 2. Since M's running time is bounded by T, M can use at most T of its random bits. If M outputs 0 for m of the 2^T strings of length T, then its probability of outputting 0 is $m/2^T$, and cannot be 1/3.

should be played. For example, in the second variant of the roshambo example above, we can get an ϵ-equilibrium for a small ϵ by attempting to simulate the uniform distribution by tossing the coin 10,000 times, and then just playing rock if 10,000 tails are tossed. Whether it is worth continuing the simulation after 10,000 tails depends on the cost of the additional computation. Finally, as we show below, there are natural classes of games where a Nash equilibrium is guaranteed to exist (see Section 5). With regard to the second point, we would argue that, in fact, it is the case that people continue to play certain strategies that they know are not optimal because of the overhead of switching to a different strategy; that is, our model captures a real-world phenomenon.

Another property of (standard) Bayesian games that does not hold for Bayesian machine games is the following. Given a Nash equilibrium $\vec{\sigma}$ in a Bayesian game, not only is σ_i a best response to $\vec{\sigma}_{-i}$, but it continues to be a best response conditional on i's type. That is, if Pr is the probability distribution on types, and $\Pr(t_i) > 0$, then $U_i(\sigma_i, \vec{\sigma}_{-i} \mid t_i) \geq U_i(\sigma_i', \vec{\sigma}_{-i} \mid t_i)$ for all strategies σ_i' for player i, where $U_i(\sigma_i, \vec{\sigma}_{-i} \mid t_i)$ is the expected utility of $\vec{\sigma}$ conditional on player i having type i. Intuitively, if player i could do better than playing σ_i if his type were t_i by playing σ_i', then σ_i would not be a best response to $\vec{\sigma}_{-i}$; we could just modify σ_i to agree with σ_i' when i's type is t_i to get a strategy that does better against $\vec{\sigma}_{-i}$ than σ_i. This is no longer true with Bayesian machine games, as the following simple example shows.

Example 2.4 (Primality guessing) Suppose that the probability on the type space assigns uniform probability to all 2^{100} odd numbers between 2^{100} and 2^{101} (represented as bit strings of length 100). Suppose that a player i wants to compute if its input (i.e., its type) is prime. Specifically, i gets a utility of 2 minus the costs of its running time (explained below) if t is prime and it outputs 1 or if t is composite and it outputs 0; on the other hand, if it outputs either 0 or 1 and gets the wrong answer, then it gets a utility of -1000. But i also has the option of "playing safe"; if i outputs 2, then i gets a utility of 1 no matter what the input is. The running-time cost is taken to be 0 if i's machine takes less than 2 units of time and otherwise is -2. We assume that outputting a constant function takes 1 unit of time. Note that although testing for primality is in polynomial time, it will take more than 2 units of time on all inputs that have positive probability. Since i's utility is independent of what other players do, i's best response is to always output 2. However, if t is actually a prime, i's best response conditional on t is to output 1; similarly, if t is not a prime, i's best response conditional on t is to output 0. The key point is that the machine that outputs i's best response conditional on a type does not do any computation; it just outputs the appropriate value.

Note that here we are strongly using the assumption that i understands the utility of outputting 0 or 1 conditional on type t. This amounts to saying that if he is playing the game conditional on t, then he has enough experience with the game to know whether t is prime. If we wanted to capture a more general setting where the player did not understand the game, even after learning t, then we could do this by considering two types (states) of nature, s_0 and s_1, where, intuitively, t is composite if the state (nature's type) is s_0 and prime if it is s_1. Of course, t is either prime or it is not. We can avoid this problem by simply having the utility of outputting 1 in s_0 or 0 in s_1 being -2 (because, intuitively, in state s_0, t is composite and in s_1 it is prime) and the utility of outputting 0 in s_0 or 1 in s_1 being 2. The relative probability of s_0 and s_1 would reflect the player i's prior probability that t is prime.

In this case, there was no uncertainty about the complexity; there was simply uncertainty about whether the type satisfied a certain property. As we have seen, we can already model the latter type of uncertainty in our framework. To model uncertainty about complexity, we simply allow the complexity function to depend on nature's type, as well as the machine and the input. We leave the straightforward details to the reader. ∎

A common criticism (see e.g., [Aumann 1985]) of Nash equilibria that use randomized strategies is that such equilibria cannot be *strict* (i.e., it cannot be the case that each player's equilibrium strategy gives a strictly better payoff than any other strategy, given the other players' strategies). This follows since any pure strategy in the support of the randomized strategy must give the same payoff as the randomized strategy. As the example below shows, this is no longer the case when considering games with computational costs.

Example 2.5 (Strict Nash equilibrium) Consider the same game as in Example 2.4, except that all machines with running time less than or equal to T have a cost of 0, and machines that take time greater than T have a cost of -2. It might very well be the case that, for some values of T, there might be a probabilistic primality testing algorithms that runs in time T and determines with high probability whether a given input x is prime or composite, whereas all deterministic algorithms take too much time. (Indeed, although deterministic polynomial-time algorithms for primality testing are known [Agrawal, Keyal, and Saxena 2004], in practice, randomized algorithms are used because they run significantly faster.) ∎

3 Machine Games with Mediators

Up to now we have assumed that the only input a machine receives is the initial type. This is appropriate in a normal-form game, but does not allow us to consider game where players can communicate with each other and (possibly) with a trusted mediator. We now extend Bayesian machine games to allow for communication. For ease of exposition, we assume that all communication passes between the players and a trusted mediator. Communication between the players is modeled by having a trusted mediator who passes along messages received from the players. Thus, we think of the players as having reliable communication channels to and from a mediator; no other communication channels are assumed to exist.

The formal definition of a Bayesian machine game with a mediator is similar in spirit to that of a Bayesian machine game, but now we assume that the machines are *interactive* Turing machines, that can also send and receive messages. We omit the formal definition of an interactive Turing machine (see, for example, [Goldreich 2001]); roughly speaking, the machines use a special tape where the message to be sent is placed and another tape where a message to be received is written. The mediator is modeled by an interactive Turing machine that we denote $\mathcal{F}$.

A *Bayesian machine game with a mediator* (or a mediated Bayesian machine game) is thus a pair $(G, \mathcal{F})$, where $G = ([m], \mathcal{M}, \Pr, \mathscr{C}_1, \ldots, \mathscr{C}_m, u_1, \ldots, u_m)$ is a Bayesian machine game (except that $\mathcal{M}$ here denotes a set of *interactive* machines) and $\mathcal{F}$ is an interactive Turing machine.

Like machines in Bayesian machine games, interactive machines in a game with a mediator take as argument a view and produce an outcome. Since what an interactive machine does can depend on the history of messages sent by the mediator, the message history (or, more precisely, that part of the message history actually read by the machine) is also part of the view. Thus, we now define a view to be a string $t; h; r$ in $\{0,1\}^*; \{0,1\}^*; \{0,1\}^*$, where, as before, t is that part of the type actually read and r is a finite bitstring representing the string of random bits actually used, and h is a finite sequence of messages received and read. Again, if $v = t; h; r$, we take $M(v)$ to be the output of M given the view.

We assume that the system proceeds in synchronous stages; a message sent by one machine to another in stage k is received by the start of stage $k+1$. More formally, following [Abraham, Dolev, Gonen, and Halpern 2006], we assume that a *stage* consists of three phases. In the first phase of a stage, each player i sends a message to the mediator, or, more precisely, player i's machine M_i computes a message to send to the mediator; machine M_i can also send an empty message, denoted

λ. In the second phase, the mediator receives the message and mediator's machine sends each player i a message in response (again, the mediator can send an empty message). In the third phase, each player i performs an action other than that of sending a message (again, it may do nothing). The messages sent and the actions taken can depend on the machine's message history (as well as its initial type).

We can now define the expected utility of a profile of interactive machines in a Bayesian machine game with a mediator. The definition is similar in spirit to the definition in Bayesian machine games, except that we must take into account the dependence of a player's actions on the message sent by the mediator. Let $\mathsf{view}_i(\vec{M}, \mathcal{F}, \vec{t}, \vec{r})$ denote the string $(t_i; h_i; r_i)$, where h_i denotes the messages received by player i if the machine profile is $\vec{M}$, the mediator uses machine $\mathcal{F}$, the type profile is $\vec{t}$, and $\vec{r}$ is the profile of random strings used by the players and the mediator. Given a mediated Bayesian machine game $G' = (G, \mathcal{F})$, we can define the random variable $u_i^{G',\vec{M}}(\vec{t}, \vec{r})$ as before, except that now $\vec{r}$ must include a random string for the mediator, and to compute the outcome and the complexity function, M_j gets as an argument $\mathsf{view}_j(\vec{M}, \mathcal{F}, \vec{t}, \vec{r})$, since this is the view that machine M_j gets in this setting. Finally, we define $U_i^{G'}(\vec{M}) = \mathbf{E}_{\Pr^+}[u_i^{G',\vec{M}}]$ as before, except that now $\Pr^+$ is a distribution on $T \times (\{0,1\}^\infty)^{m+1}$ rather than $T \times (\{0,1\}^\infty)^m$, since we must include a random string for the mediator as well as the players' machines.

We can define Nash equilibrium as in Bayesian machine games; we leave the details to the reader.

As the following example shows, the *revelation principle* no longer holds once we take computation into account.

Example 3.1 (Failure of revelation principle) The revelation principle is one of the fundamental principles in traditional implementation theory. A specific instance of it [Myerson 1979; Forges 1986] stipulates that for every Nash equilibrium in a mediated games $(\mathcal{G}, \mathcal{F})$, there exists a different mediator $\mathcal{F}'$ such that it is a Nash equilibrium for the players to *truthfully* report their type to the mediator and then perform the action suggested by the mediator. As we demonstrate, this principle no longer holds when we take computation into account (a similar point is made by Conitzer and Sandholm [2004], although they do not use our formal model). The intuition is simple: truthfully reporting your type will not be an equilibrium if it is too "expensive" to send the whole type to the mediator. For a naive counter example, consider a game where a player get utility 1 whenever its complexity is 0 (i.e., the player uses the strategy $\perp$) and utility 0 otherwise. Clearly, in this game it can never be an equilibrium to truthfully report your type to any mediator. This example is degenerate as the players actually never use the mediator. In the following example, we consider a game with a Nash equilibrium where the players use the mediator.

Consider a 2-player game where each player's type is an n-bit number. The type space consists of all pairs of n-bit numbers that either are the same, or that differ in all but at most k places, where $k \ll n$. The players receive a utility of 1 if they can guess correctly whether their types are the same or not, while having communicated less than $k + 2$ bits; otherwise, they receive a utility of 0. Consider a mediator that, upon receiving $k + 1$ bits from each of the players, tells the players whether the bit sequences are identical. With such a mediator it is an equilibrium for the players to provide the first $k + 1$ bits of their input and then output whatever the mediator tells them. However, providing the full type is always too expensive (and can thus never be a Nash equilibrium), no matter what mediator the players have access to. ∎

Repeated games and, more generally, arbitrary extensive-form games (i.e., games defined by game trees) can be viewed as special cases of games with mediators, except that now we allow the utility to depend on the messages sent to the mediator (since we view these as encoding the actions taken in the game). In more detail, we capture an extensive-form machine game $G' = (G, \mathcal{N})$

by simply viewing nature as a mediator $\mathcal{N}$; we allow the mediator to be a function of the type of nature, and the utility functions to take into account the messages sent by the players to the mediator (i.e., talk is not necessarily "cheap"), and also the random coins used by the mediator. In other words, we assume without loss of generality that all communication and signals sent to a player are sent through the mediator, and that all actions taken by a player are sent as messages to the mediator. Thus, the utility of player i, $u_i(\vec{t}, h, \vec{c})$, is now a function of the type profile $\vec{t}$, the view h of nature (i.e., the messages received and random coins tossed by the mediator), and the complexity profile $\vec{c}$. We leave the details of the formal definition to the reader.

4 Computational Explanations for Observed Behavior

In this section we show several examples where our framework can be used to give simple (and, arguably, natural) explanations for experimentally observed behavior in classical games where traditional game-theoretic solution concepts fail. For instance, we show that in a variant of *finitely repeated prisoner's dilemma (FRPD)* where players are charged for the use of memory, the strategy *tit for tat*—which does exceedingly well in experiments [Axelrod 1984]—is also a Nash equilibrium, whereas it is dominated by defecting in the classical sense. (Intuitively, this follows from the facts that (1) any profitable deviation from tit for tat requires the use of memory, and (2) if future utility is discounted, then for sufficiently long games, the potential gain of deviating becomes smaller than the cost of memory.) Similarly, the imperfect randomization observed in real-life events where randomization helps (e.g., penalty shots in soccer or serving in tennis) can be explained by considering computational costs; as long as a sequence appears random relative to the computation that an opponent is willing to put in to make predictions about the sequence, that is good enough. Interestingly, the same explanation applies to the computational theory of *pseudorandomness*. A sequence is pseudorandom if it passes all polynomial-time tests for randomness. There is no need to use a sequence that is more random than a pseudorandom sequence when playing against an opponent for whom super-polynomial computation is too expensive. In a companion paper [Halpern and Pass 2010], where we apply our framework to decision theory, we show how other well-known "anomalous" behaviors, such as the *status quo* bias [Samuelson and Zeckhauser 1998] and the *first-impressions-matter* bias [Rabin 1998] can be explained by viewing computation as a costly resource. Many of these behaviors have been given various cognitive explanations (e.g., using models of the brain [Mullainathan 2002] or using psychology [Rabin 1998]). Some of these explanations can be viewed as being based in part on considerations of computational cost. As shown by our results, we do not need to resort to complicated assumptions about the brain, or psychology, in order to provide such explanations: extremely simple (and plausible) assumptions about the cost of computation suffice to explain, at least at a qualitative level, the observed behavior. (See [Halpern and Pass 2010] and Example 4.4 for further discussion of these points.)

Example 4.1 (Finitely repeated prisoner's dilemma) Recall that in the prisoner's dilemma, there are two prisoners, who can choose to either cooperate or defect. As described in the table below, if they both cooperate, they both get 3; if they both defect, then both get -3; if one defects and the other cooperates, the defector gets 5 and the cooperator gets -5. (Intuitively, the cooperator stays silent, while the defector "rats out" his partner. If they both rat each other out, they both go to jail.)

It is easy to see that defecting dominates cooperating: no matter what the other player does, a player is better off defecting than cooperating. Thus, "rational" players should defect. And, indeed, (D, D) is the only Nash equilibrium of this game. Although (C, C) gives both players a better payoff than (D, D), this is not an equilibrium.

	C	D
C	$(3,3)$	$(-5,5)$
D	$(5,-5)$	$(-3,-3)$

Now consider finitely repeated prisoner's dilemma (FRPD), where prisoner's dilemma is played for some fixed number N of rounds. The only Nash equilibrium is to always defect; this can be seen by a backwards induction argument. (The last round is like the one-shot game, so both players should defect; given that they are both defecting at the last round, they should both defect at the second-last round; and so on.) This seems quite unreasonable. And, indeed, in experiments, people do not always defect [Axelrod 1984]. In fact, quite often they cooperate throughout the game. Are they irrational? It is hard to call this irrational behavior, given that the "irrational" players do much better than supposedly rational players who always defect.

There have been many attempts to explain cooperation in FRPD in the literature; see, for example, [Kreps, Milgrom, Roberts, and Wilson 1982; Neyman 1985; Papadimitriou and Yannakakis 1994]. In particular, [Neyman 1985; Papadimitriou and Yannakakis 1994] demonstrate that if players are restricted to using a finite automaton with bounded complexity, then there exist equilibria that allow for cooperation. However, the strategies used in those equilibria are quite complex, and require the use of large automata;[2] as a consequence this approach does not seem to provide a satisfactory explanation as to why people choose to cooperate.

By using our framework, we can provide a straightforward explanation. Consider the *tit for tat* strategy, which proceeds as follows: a player cooperates at the first round, and then at round $m+1$, does whatever his opponent did at round m. Thus, if the opponent cooperated at the previous round, then you reward him by continuing to cooperate; if he defected at the previous round, you punish him by defecting. If both players play tit for tat, then they cooperate throughout the game. Interestingly, tit for tat does exceedingly well in FRPD tournaments, where computer programs play each other [Axelrod 1984].

Now consider a machine-game version of FRPD, where at each round the player receive as signal the move of the opponent in the previous rounds before they choose their action. In such a game, tit for tat is a simple program, which needs no memory (i.e., the TM uses only one state). Suppose that we charge even a modest amount α for memory usage (i.e., machines that use memory get a penalty of at least α, whereas memoryless machines get no penalty), that there is a discount factor δ, with $0.5 < \delta < 1$, so that if the player gets a reward of r_m in round m, his total reward over the whole N-round game (excluding the complexity penalty) is taken to be $\sum_{m=1}^{N} \delta^m r_m$, that $\alpha \geq 2\delta^N$, and that $N > 2$. In this case, it will be a Nash equilibrium for both players to play tit for tat. Intuitively, no matter what the cost of memory is (as long as it is positive), for a sufficiently long game, tit for tat is a Nash equilibrium.

To see this, note that the best response to tit for tat is to play tit for tat up to but not including the last round, and then to defect. But following this strategy requires the player to keep track of the round number, which requires the use of extra memory. The extra gain of 2 achieved by defecting at the last round will not be worth the cost of keeping track of the round number as long as $\alpha \geq 2\delta^N$; thus no strategy that use some memory can do better. It remains to argue that no memoryless strategy (even a randomized memoryless strategy) can do better against tit for tat. Note that any strategy that defects for the first time at round $k < N$ does at least $6\delta^{k+1} - 2\delta^k$ worse than tit for tat. It gains 2 at round k (for a discounted utility of $2\delta^k$), but loses at least 6 relative

[2]The idea behind these equilibria is to force players to remember a short history of the game, during which players perform random actions; this requires the use of many states.

to tit for tat in the next round, for a discounted utility of $6\delta^{k+1}$. From that point on the best response is to either continue defecting (which at each round leads to a loss of 6), or cooperating until the last round and then defecting (which leads to an additional loss of 2 in round $k+1$, but a gain of 2 in round N). Thus, any strategy that defects at round $k < N$ does at least $6\delta^{k+1} - 2\delta^k$ worse than tit for tat.

A strategy that defects at the last round gains $2\delta^N$ relative to tit for tat. Since $N > 2$, the probability that a memoryless strategy defects at round $N-1$ or earlier is at least as high as the probability that it defects for the first time at round N. (We require that $N > 2$ to make sure that there exist some round $k < N$ where the strategy is run on input C.) It follows that any stateless strategy that defects for the first time in the last round with probability p in expectation gains at most $p(2\delta^N - (6\delta^N - 2\delta^{N-1})) = p\delta^{N-1}(2 - 4\delta)$, which is negative when $\delta > 0.5$. Thus, when $\alpha \geq 2\delta^N$, $N > 2$, and $\delta > 0.5$, tit for tat is a Nash equilibrium in FRPD. (However, also note that depending on the cost of memory, tit for tat may *not* be a Nash equilibrium for sufficiently short games.)

The argument above can be extended to show that tit for tat is a Nash equilibrium even if there is also a charge for randomness or computation, as long as there is no computational charge for machines as "simple" as tit for tat; this follows since adding such extra charges can only make things worse for strategies other than tit for tat. Also note that even if only one player is charged for memory, and memory is free for the other player, then there is a Nash equilibrium where the bounded player plays tit for tat, while the other player plays the best response of cooperating up to but not including the last round of the game, and then defecting in the last round. (A similar argument works without assuming a discount factor (or, equivalently, taking $\delta = 1$) if we assume that the cost of memory increases unboundedly with N.)

Note that the assumption of a cost for remembering the round number can be tested by running experiments with a variant of the FRPD where players have access to a counter showing how many rounds remain in the game. If players' behavior changes in the presence of such as counter, this could be interpreted as suggesting a cognitive cost for remembering the round number. In fact, competitive swimmers have, and make use of, counters telling them how many laps remain in the race.[3] ∎

Example 4.2 (Biases in randomization) An interesting study by Walker and Wooders [2001] shows that even in professional sports competitions (when large sums of money are at play), players randomize badly. For instance, in tennis serving, we can view the server as choosing between two actions: serving left or serving right, while the receiver attempts to predict the action of the server. However, as observed by Walker and Wooders [2001], while players do indeed randomize, their choices are not made independently at each step. In particular, they tend to switch from one action to another too often.

These observations can again be explained by assuming a cost for computation. Suppose that the server could actually randomize well if he wanted to, but that there is some (possibly small) cost for doing this. For instance, the server may make use of features of nature to generate her randomness, but this might entail a loss in concentration. Similarly, if the server indeed uses a weak method of randomization, the receiver can try to "break" it, but this again requires some computational effort (and again a loss of concentration). So, under reasonable assumptions, there is an equilibrium where the server uses only weak randomization, while the receiver uses only a computationally-weak predictor (which can be fooled by the randomization used by the server).

Indeed, the usage of such methods is prevalent in the computer science literature: the definition of *pseudo-randomness* [Blum and Micali 1984; Yao 1982] recognizes that a sequence of numbers

[3] We thank Rachel Kranton for this observation.

needs to be only "random-looking" with respect to a particular class of computationally resource-bounded observers. It is also well-known that computationally-weak classes of observers can be easily fooled. For example, a polynomial-size constant-depth circuit (a quite weak model of computation) cannot distinguish a random sequence of bits whose parity is 1 from a random sequence with parity 0 [Håstad 1989].

Let us elaborate. We first recall the cryptographic (complexity-theoretic) definition of a *pseudorandom generator* [Blum and Micali 1984; Yao 1982]. Intuitively, a pseudorandom generator (PRG) is a deterministic function that takes a short random "seed", and expands it to a longer string that "looks random" to a computationally bounded observer. More precisely, call a TM T-bounded if both its size and running time are bounded by T; we now require that no T-bounded observer can guess the next bit of the sequence output by the PRG with probability significantly better than $1/2$, given any prefix of it.[4]

More formally, suppose that $g : \{0,1\}^{\ell_0} \to \{0,1\}^{\ell}$. (Intuitively, g is a pseudorandom generator, and ℓ_0 is much smaller than ℓ.) Given a TM M, define the predicate $\mathsf{predict}_M^{g,i} : \{0,1\}^{\ell_0} \times \{0,1\}^{\infty} \to \{0,1\}$ by taking $\mathsf{predict}_M^{g,i}(s,r) = 1$ iff $M(g(s)_{0\to i}; r) = g(s)_{i+1}$, where $X_{0\to i}$ denotes the first i bits of X; that is, $\mathsf{predict}_M^{g,i}(s,r) = 1$ iff M, using the random string, r can guess the $(i+1)$st bit of $g(s)$ given only the first i bits of $g(s)$. Let $\Pr_l$ be the uniform distribution over l-bit strings.

Definition 4.3 *The function $g : \{0,1\}^{\ell_0} \to \{0,1\}^{\ell}$ is a (T,ϵ)-*pseudorandom generator (PRG) *if $\ell > \ell_0$ and, for all T-bounded TM machines M and all $i \in \{0,1,\ldots,\ell-1\}$, $\Pr^{+}_{\ell_0}[\mathsf{predict}_M^{g,i} = 1] \leq \frac{1}{2} + \epsilon$.*

The existence of efficient pseudorandom generators has been established under a variety of number-theoretic conjectures (e.g., the hardness dof factoring products of large primes, or the hardness of the discrete logarithm problem) [Blum and Micali 1984; **?**].

Now consider the following simplified game representation of the tennis-serving scenario. The game consists of ℓ rounds, where in round i, player 1 picks a bit x_i as its action (intuitively, this is meant to model the server choosing where to serve) and player 2 may either pick a bit g_i ("a guess for x_i") or output the empty string (for "no guess"). The utilities are specified as follows. For each round i where player 2 outputs a correct guess (i.e., $g_i = x_i$), player 2 gets a score of 1 and player 1 gets a score of -1; when the guess is incorrect, player 1 gets a score of 1 and player 2 a score of -1; if player 2 outputs the empty string, both players get a score of 0. (For simplicity, we are assuming that if player 2 does not try to guess where player 1 will serve, then he will concentrate, and the players are equally likely to do well. In practice, the server has an advantage, but this is irrelevant to the point we are trying to make.) The final utility in the game is the total score the players accumulate, with a penalty for the computational resources used. For simplicity, player 1 receives a "huge" penalty $p > \ell$ if it uses more than ℓ_0 random coins (that is, if it looks at more than the first ℓ_0 bits of the random string r), and otherwise gets a penalty of 0; player 2 receives 0 penalty if it "does nothing" (i.e., always just outputs the empty string), a "small" penalty $p_0 = \epsilon\ell + 1$ if it uses a machine that does not always output the empty string, but still is T-bounded, and a huge penalty p if it uses a machine that is not T-bounded. In such a scenario, it is easy to see that, assuming the existence of a (T',ϵ)-PRG $g : \{0,1\}^{\ell_0} \to \{0,1\}^{\ell}$, where T' is slightly larger than T (the exact choice of T' is explained below), it is a NE for player i to sample ℓ_0 random bits s, compute $x = g(s)$, and in round i to output x_i, and for player 2 to "do nothing". By the security of the PRG g, we have that, for every round i, player 2 can guess x_i with probability at most $\frac{1}{2} + \epsilon$ if it uses a machine that is T-bounded, but the potential ϵ advantage is eaten up by the "small"

[4] As shown by Yao [1982], this condition is equivalent to saying that no computationally bounded "distinguisher" can tell apart the output of the generator from a truly random string.

penalty of p_0 for attempting a guess. (The reason we require g to be secure for $T' > T$-bounded machine is a rather technical one: there might be a small overhead in turning a machine M playing the role of player 2 in the game into a machine M' violating the security of the PRG. Given an input x of length k and random string r, M' simply outputs what M would output if M had view x (that is, if, in consecutive rounds, M saw $x_1, \ldots x_k$) and random string r. Although running M takes at most T steps, constructing the view may take a few extra steps, so we need to have $T' > T$.) As we mentioned, under standard cryptographic assumptions, there are computationally efficient PRGs, so the same strategies remain a NE even if we add a "huge" penalty p to player 1 unless it uses a T-bounded strategy. ∎

Example 4.4 (Biases in decision theory) There are several examples of single-agent decision theory problem where experimentally observed behavior does not correspond to what is predicted by standard models. For instance, psychologists have observed systematic biases in the way that individuals update their beliefs as new information is received (see [Rabin 1998] for a survey). In particular, a *first-impressions-matter* bias has been observed: individuals put too much weight on initial signals and less weight on later signals. As they become more convinced that their beliefs are correct, many individuals even seem to simply ignore all information once they reach a confidence threshold. Other examples of such phenomena are *belief polarization* [Lord, Ross, and Lepper 1979]—two people, hearing the same information (but with possibly different prior beliefs) can end up with diametrically opposed conclusions, and the *status quo* bias [Samuelson and Zeckhauser 1998]—people are much more likely to stick with what they already have. In a companion paper [Halpern and Pass 2010], where we apply our framework to decision theory, we show how these biases can be explained by considering computation as a costly resource.

We here briefly show how the first-impressions-matter bias can be explained by considering a small cost for "absorbing" new information. Consider the following simple game (which is very similar to one studied in [Mullainathan 2002; Wilson 2002]). The state of nature is a bit b which is 1 with probability 1/2. An agent receives as his type a sequence of independent samples $s_1, s_2, \ldots, s_n$ where $s_i = b$ with probability $\rho > 1/2$. The samples corresponds to signals the agents receive about b. An agent is supposed to output a guess b' for the bit b. If the guess is correct, he receives $1 - mc$ as utility, and $-mc$ otherwise, where m is the number of bits of the type he read, and c is the cost of reading a single bit (c should be thought of the cost of absorbing/interpreting information). It seems reasonable to assume that $c > 0$; signals usually require some effort to decode (such as reading a newspaper article, or attentively watching a movie). If $c > 0$, it easily follows by the Chernoff bound that after reading a certain (fixed) number of signals $s_1, \ldots, s_i$, the agents will have a sufficiently good estimate of ρ that the marginal cost of reading one extra signal s_{i+1} is higher than the expected gain of finding out the value of s_{i+1}. That is, after processing a certain number of signals, agents will eventually disregard all future signals and base their output guess only on the initial sequence. In fact, doing so stricly dominates reading more signals. ∎

5 Sufficient Conditions for the Existence of Nash Equilibrium

As Example 2.3 shows, Nash equilibrium does not always exist in machine games. The complexity function in this example charged for randomization. Our goal in this section is to show that this is essentially the reason that Nash equilibrium did not exist; if randomization were free (as it is, for example, in the model of [Ben-Sasson, Kalai, and Kalai 2007]), then Nash equilibrium would always exist.

This result turns out to be surprisingly subtle. To prove it, we first consider machine games where *all* computation is free, that is, the utility of a player depends only on the type and action profiles (and not the complexity profiles). For ease of notation, we here restrict attention to

Bayesian machine games, but it can be easily seen that our results apply also to extensive-form machine games. Formally, a machine game $G = ([m], \mathcal{M}, \Pr, T, \vec{\mathscr{C}}, \vec{u})$ is *computationally cheap* if $\vec{u}$ depends only on the type and action profiles, that is, if there exists $\vec{u'}$ such that $\vec{u}(\vec{t}, \vec{a}, \vec{c}) = \vec{u'}(\vec{t}, \vec{a})$ for all $\vec{t}, \vec{a}, \vec{c}$.

We would like to show that every computationally cheap Bayesian machine game has a Nash equilibrium. But this is too much to hope for. The first problem is that the game may have infinitely many possible actions, and may not be compact in any reasonable topology. This problem is easily solved; we will simply require that the type space and the set of possible actions be finite. Given a *bounding function* $B : \mathrm{N} \to \mathrm{N}$, a *$B$-bounded Turing machine* M is one such that for each $x \in \{0,1\}^n$, the output of $M(x)$ has length at most $B(n)$. If we restrict our attention to games with a finite type space where only B-bounded machines can be used for some bounding function B, then we are guaranteed to have only finitely many types and actions.

With this restriction, since we do not charge for computation in a computationally cheap game, it may seem that this result should follow trivially from the fact that every finite game has a Nash equilibrium. But this is false. The problem is that the game itself might involve non-computable features, so we cannot hope that that a Turing machine will be able to play a Nash equilibrium, even if it exists.

Recall that a real number r is *computable* [Turing 1937] if there exists a Turing machine that on input n outputs a number r' such that $|r - r'| < 2^{-n}$. A game $G = ([m], \mathcal{M}, \Pr, T, \vec{\mathscr{C}}, \vec{u})$ is *computable* if (1) for every $\vec{t} \in T$, $\Pr[\vec{t}]$ is computable, and (2) for every $\vec{t}, \vec{a}, \vec{c}$, $u(\vec{t}, \vec{a}, \vec{c})$ is computable. As we now show, every computationally cheap *computable* Bayesian machine game has a Nash equilibrium. Even this result is not immediate. Although the game itself is computable, a priori, there may not be a computable Nash equilibrium. Moreover, even if there is, a Turing machine may not be able to simulate it. Our proof deals with both of these problems.

To deal with the first problem, we follow lines similar to those of Lipton and Markakis [2004], who used the Tarski-Seidenberg *transfer principle* [Tarski 1951] to prove the existence of *algebraic* Nash equilibria in finite normal-form games with integer-valued utilities. (Similar techniques were also used by Papadimitriou and Roughgarden [2005].) We briefly review the relevant details here.

Definition 5.1 *An ordered field R is a* real closed field *if every positive element x is a square (i.e., there exists a $y \in R$ such that $y^2 = x$), and every univariate polynomial of odd degree with coefficients in R has a root in R.*

Of course, the real numbers are a real closed field. It is not hard to check that the computable numbers are a real closed field as well.

Theorem 5.2 (Tarski-Seidenberg [Tarski 1951]) *Let R and R' be real closed fields such that $R \subseteq R'$, and let $\bar{P}$ be a finite set of (multivariate) polynomial inequalities with coefficients in R. Then $\bar{P}$ has a solution in R if and only if it has a solution in R'.*

With this background, we can state and prove the theorem.

Theorem 5.3 *If T is a finite type space, B is a bounding function, $\mathcal{M}$ is a set of B-bounded machines, and $G = ([m], \mathcal{M}, \Pr, \vec{\mathscr{C}}, \vec{u})$ is a computable, computationally cheap Bayesian machine game, then there exists a Nash equilibrium in G.*

Proof: Note that since in G, (1) the type set is finite, (2) the machine set contains only machines with bounded output length, and thus the action set A is finite, and (3) computation is free, there exists a finite (standard) Bayesian game G' with the same type space, action space, and utility functions as G. Thus, G' has a Nash equilibrium.

Although G' has a Nash equilibrium, some equilibria of G' might not be implementable by a randomized Turing machine; indeed, Nash [1951] showed that even if all utilities are rational, there exist normal-form games where all Nash equilibria involve mixtures over actions with irrational probabilities. To deal with this problem we use the transfer principle.

Let R' be the real numbers and R be the computable numbers. Clearly $R \subset R'$. We use the approach of Lipton and Markakis to show that a Nash equilibrium in G' must be the solution to a set of polynomial inequalities with coefficients in R (i.e., with computable coefficients). Then, by combining the Tarski-Seidenberg transfer principle with the fact that G' has a Nash equilibrium, it follows that there is a computable Nash equilibrium.

The polynomial equations characterizing the Nash equilibria of G' are easy to characterize. By definition, $\vec{\sigma}$ is a Nash equilibrium of G' if and only if (1) for each player i, each type $t_i \in T_i$, and $a_i \in A_i$, $\sigma(t_i, a_i) \geq 0$, (2) for each player i and t_i, $\sum_{a_i \in A_i} \sigma(t_i, a_i) = 1$, and (3) for each player i $t_i \in T$, and action $a'_i \in A$,

$$\sum_{\vec{t}_{-i} \in T_{-i}} \sum_{\vec{a} \in A} \Pr(\vec{t}) u'_i(\vec{t}, \vec{a}) \prod_{j \in [m]} \sigma_j(t_j, a_j) \geq \sum_{\vec{t}_{-i} \in T_{-i}} \sum_{\vec{a}_{-i} \in A_{-i}} \Pr(\vec{t}) u'_i(\vec{t}, (a'_i, \vec{a}_{-i})) \prod_{j \in [m] \setminus i} \sigma_j(t_j, a_j).$$

Here we are using the fact that a Nash equilibrium must continue to be a Nash equilibrium conditional on each type.

Let P be the set of polynomial equations that result by replacing $\sigma_j(t_j, a_j)$ by the variable x_{j,t_j,a_j}. Since both the type set and action set is finite, and since both the type distribution and utilities are computable, this is a finite set of polynomial inequalities with computable coefficients, whose solutions are the Nash equilibria of G'. It now follows from the transfer theorem that G' has a Nash equilibrium where all the probabilities $\sigma_i(t_i, a_i)$ are computable.

It remains only to show that this equilibrium can be implemented by a randomized Turing machine. We show that, for each player i, and each type t_i, there exists a randomized machine that samples according to the distribution $\sigma_i(t_i)$; since the type set is finite, this implies that there exists a machine that implements the strategy σ_i.

Let $a_1, \ldots, a_N$ denote the actions for player i, and let $0 = s_0 \leq s_1 \leq \ldots \leq s_N = 1$ be a sequence of numbers such that $\sigma_i(t_i, a_j) = s_j - s_{j-1}$. Note that since σ is computable, each number s_j is computable too. That means that there exists a machine that, on input n, computes an approximation $\tilde{s}_{n_j}$ to s_j, such that $\tilde{s}_{n_j} - 2^{-n} \leq s_j \leq \tilde{s}_{n_j} + 2^{-n}$. Now consider the machine $M_i^{t_i}$ that proceeds as follows. The machine constructs a binary decimal $.r_1 r_2 r_3 \ldots$ bit by bit. After the nth step of the construction, the machine checks if the prefix of the decimal constructed thus far $(.r_1 \ldots r_n)$ already determines in which interval $(s_k, s_{k+1}]$ the full decimal is in. (Since $s_0, \ldots, s_N$ are computable, it can do this by approximating each one to within 2^{-n}.) With probability 1, after a finite number of steps, the decimal expansion will be known to lie in a unique interval $(s_k, s_{k+1}]$. When this happens, action a_k is performed. $\square$

We now want to prove that a Nash equilibrium is guaranteed to exist provided that randomization is free. Thus, we assume that we start with a finite set $\mathcal{M}_0$ of *deterministic* Turing machines and a finite set T of types. $\mathcal{M}$ is the *computable convex closure of* $\mathcal{M}_0$ if $\mathcal{M}$ consists of machines M that, on input (t, r), first perform some computation that depends only on the random string r and not on the type t, that with probability 1, after some finite time and after reading a finite prefix r_1 of r, choose a machine $M' \in \mathcal{M}_0$, then run M' on input (t, r_2), where r_2 is the remaining part of the random tape r. Intuitively, M is randomizing over the machines in $\mathcal{M}_0$. Since machines in $\mathcal{M}_0$ are deterministic, it is easy to see that there must be some B such that all the machines in $\mathcal{M}$ are B-bounded. *Randomization is free* in a machine game $G = ([m], \mathcal{M}, \Pr, T, \vec{\mathscr{C}}, \vec{u})$ where $\mathcal{M}$ is the computable convex closure of $\mathcal{M}_0$ if $\mathscr{C}_i(M, t, r)$ is $\mathscr{C}_i(M', t, r_2)$ (using the notation from above).

Theorem 5.4 *If $\mathcal{M}$ is the computable convex closure of some finite set $\mathcal{M}_0$ of deterministic Turing machines, T is a finite type space, and $G = ([m], \mathcal{M}, T, Pr, \vec{\mathscr{C}}, \vec{u})$ is a game where randomization is free, then there exists a Nash equilibrium in G.*

Proof Sketch: First consider the normal-form game where the agents choose a machine in $\mathcal{M}_0$, and the payoff of player i if $\vec{M}$ is chosen is the expected payoff in G (where the expectation is taken with respect to the probability Pr on T). By Nash's theorem, it follows that there exists a mixed strategy Nash equilibrium in this game. Using the same argument as in the proof of Theorem 5.3, it follows that there exists a machine in $\mathcal{M}$ that samples according to the mixed distribution over machines, as long as the game is computable (i.e., the type distribution and utilities are computable) and the type and action spaces finite. (The fact that the action space is finite again follows from the fact that type space and $\mathcal{M}_0$ are finite and that all machines in $\mathcal{M}_0$ are deterministic.) The desired result follows. ∎

We remark that if we take Aumann's [1987] view of a mixed-strategy equilibrium as representing an equilibrium in players' beliefs—that is, each player is actually using a deterministic strategy in $\mathcal{M}_0$, and the probability that player i plays a strategy $M' \in \mathcal{M}_0$ in equilibrium represents the other players' beliefs about the probability that M' will be played—then we can justify randomization being free, since players are not actually randomizing. The fact that the randomization is computable here amounts to the assumption that players' beliefs are computable (a requirement advocated by Megiddo [1989]) and that the population players are chosen from can be sampled by a Turing machine. More generally, there may be settings where randomization devices are essentially freely available (although, even then, it may not be so easy to create an arbitrary computable distribution).

Theorem 5.4 shows that if randomization is free, a Nash equilibrium in machine games is guaranteed to exist. We can generalize this argument to show that, to guarantee the existence of ϵ-Nash equilibrium (for arbitrarily small ϵ), it is enough to assume that "polynomial-time" randomization is free. Lipton, Markakis and Mehta [2003] show that every finite game with action space A has an ϵ-Nash equilibrium with support on only $poly(\log |A| + 1/\epsilon)$ actions; furthermore, the probability of each action is a rational number of length $poly(\log |A| + 1/\epsilon)$. In our setting, it follows that there exists an ϵ-Nash equilibrium where the randomization can be computed by a Turing machine with size and running-time bounded by size $O(\log |\mathcal{M}'| + 1/\epsilon)$. We omit the details here.

6 Applications to Cryptography

Our focus in this paper has been on game-theoretic questions. In a companion paper [Halpern and Pass 2009a], we apply our framework to cryptography. More precisely, we show how to use it to provide a game-theoretic definition of cryptographic protocol security. We outline this approach below, and refer the reader to [Halpern and Pass 2009a] for further details.

It is often simpler to design and analyze mechanisms when assuming that players have access to a trusted mediator through which they can communicate. However, such a trusted mediator can be hard to find. A central question in cryptography is investigating under what circumstances mediators can be replaced—or *implemented*—by simple "unmediated" communication between the players.

The cryptographic notion of a *secure computation* [Goldreich, Micali, and Wigderson 1986] considers two types of players: *honest* players and *malicious* players. Honest players are assumed to faithfully execute the prescribed protocol using their intended input; malicious players, on the other hand, are assumed to do anything in their power to undermine the security of honest players.

Roughly speaking, a protocol Π is said to securely implement the mediator $\mathcal{F}$ if (1) the malicious players cannot influence the output of the communication phase any more than they could have by communicating directly with the mediator (*correctness*); and (2) the malicious players cannot "learn" more than what can be efficiently computed from only the output of mediator (*privacy*). These properties are formalized through the *zero-knowledge simulation paradigm* [Goldwasser, Micali, and Rackoff 1989]: roughly, we require that any "harm" done by an adversary in the protocol execution could be simulated by a polynomially-bounded Turing machine, called the *simulator*, that communicates only with the mediator.

There has also been work on implementation of mediators in the game-theory literature. The traditional game-theoretic notion of implementation (see [Forges 1986; Forges 1990]) does not explicitly consider properties such as privacy and correctness, but instead requires that the implementation preserve a given Nash equilibrium of the mediated game. Roughly speaking, the game-theoretic notion of implementation says that a strategy profile $\vec{\sigma}$ implements a mediator $\mathcal{F}$ if, as long as it is a Nash equilibrium for the players to tell the mediator their type and output what the mediator recommends, then $\vec{\sigma}$ is a Nash equilibrium in the "cheap talk" game (where the players just talk to each other, rather than talking to a mediator) that has the same distribution over outputs as when the players talk to the mediator. In other words, whenever a set of parties have incentives to tell the mediator their inputs, they also have incentives to honestly use $\vec{\sigma}$ using the same inputs, and get the same distribution over outputs in both cases. The key differences between the notions are that the game-theoretic notion does not consider privacy issues or computational efficiency, and the cryptographic notion does not consider incentives; the game-theoretic notion talks about preserving Nash equilibria (which cannot be done in the cryptographic notion, since there are no incentives), while the cryptographic notion talks about security against malicious adversaries.

As we show in [Halpern and Pass 2009a], the cryptographic definitions can be understood game theoretically, provided we bring complexity into the picture. This lets us capture intuitions such as that a player might not want to execute a protocol if doing so is too computationally expensive, or that a player i might want to change its input if executing the protocol gives some other player j a computational advantage.

Roughly speaking, we say that Π implements a mediator $\mathcal{F}$ if for *all* games G—including games where computation is costly—that use $\mathcal{F}$ for which (the utilities in G are such that) it is an equilibrium for the players to truthfully tell $\mathcal{F}$ their inputs, running Π on the same set of inputs (and with the same utility functions) is also an equilibrium, and produces the same distribution over outputs as $\mathcal{F}$.[5]

Note that by requiring the implementation to work for all games, not only do we ensure that players have proper incentives to execute protocols with their intended input, even if they consider computation a costly resource, but we get the privacy and correctness requirements "for free". For suppose that, when using Π, some information about i's input is revealed to j. We consider a zero-sum game G where a player j gains some significant utility by having this information. In this game, i will not want to use Π. However, our notion of implementation requires that, even with the utilities in G, i should want to use Π if i is willing to use the mediator $\mathcal{F}$. (This argument depends on the fact that we consider games where computation is costly; the fact that j gains information about i's input may mean that j can do some computation faster with this information than without it.) As a consequence, our definition gives a relatively simple (and strong) way of formalizing the security of protocols, relying only on basic notions from game theory. Perhaps surprisingly, as we show in [Halpern and Pass 2009a], under weak restrictions on the utility functions of the

[5]While the definitions of implementation in the game-theory literature (e.g., [Forges 1986; Forges 1990]) do not stress the uniformity of the implementation—that is, the fact that it works for all games—the implementations provided are in fact uniform in this sense.

players (essentially, that players never prefer to compute more), our notion of implementation is equivalent to a variant of the cryptographic notion of secure computation. This result shows that the two approaches used for dealing with "deviating" players in two different communities—*Nash equilibrium* in game theory, and *zero-knowledge "simulation"* in cryptography—are intimately connected; indeed, they are essentially equivalent in the context of implementing mediators, once we take the cost of computation into account. Moreover, thinking in game-theoretic terms suggests useful generalizations of the standard cryptographic definition; for example, by restricting the class of utility functions so that players strictly prefer to compute less, or players prefer not to get caught "cheating". See [Halpern and Pass 2009a] for further discussion of this issue.

7 Conclusion

We have defined a general approach to taking computation into account in game theory that subsumes previous approaches. This opens the door to a number of exciting research directions. We briefly describe a few here:

- In our framework we are assuming that the agents understand the costs associated with each Turing machine. That is, they do not have to do any "exploration" to compute the costs. As mentioned, we can model uncertainty regarding complexity by letting the complexity function also take the state of nature as input. However, even if we do this, we are assuming that agents can compute the probability of (or, at least, are willing to assign a probability to) events like "TM M will halt in 10,000 steps" or "the output of TM M solves the problem I am interested in on this input". But calculating such probabilities itself involves computation, which might be costly. Similarly, we do not charge the players for computing which machine is the best one to use in a given setting, or for computing the utility of a given machine profile. It would be relatively straightforward to extend our framework so that the TMs computed probabilities and utilities, as well as actions; this would allow us to "charge" for the cost of computing probabilities and utilities. However, once we do this, we need to think about what counts as an "optimal" decision if the DM does not have a probability and utility, or has a probability only on a coarse space.

  A related problem is that we have assumed that the players have all options available "for free". That is, the players choose among a set of TMs which is given *ex ante*, and does not change. But, in practice, it may be difficult to generate alternatives. (Think of a chess player trying to generate interesting lines of play, for example.) Again, we can imagine charging for generating such alternatives. One approach we are currently exploring for dealing with this cluster of problems is to assume that the DM starts with a small set of TMs that he understands (by which we mean that he understands the behavior of the TM, and has a good estimate of its running time). One of the TMs he can choose involves "thinking"; the result of such thinking is to perhaps produce further alternatives (i.e., more TMs that he can choose among), or to give a more refined or accurate estimate of the probabilities, utilities, and complexities for the TMs that he is currently aware of. If the player believes that he has already generated enough good alternatives, he may decide to stop thinking, and act. But, in general, he can go back to thinking at any time. In any case, in such a framework, it makes sense to talk about a player making a best response at all times, even as he is generating for alternatives. These ideas are clearly related to work on awareness in games (see, e.g., [Heifetz, Meier, and Schipper 2007; Halpern and Rêgo 2006]); we are planning to explore the connection.

- In a companion paper [Halpern and Pass 2010], we apply our framework to decision theory.

We show that the approach can be used to explain the status quo bias [Samuelson and Zeckhauser 1998], belief polarization [Lord, Ross, and Lepper 1979], and other biases in information processing. Moreover, we use the framework to define two extensions of the standard notion of *value of information*: *value of computational information* and *value of conversation*. Recall that value of information is meant to be a measure of how much a decision maker (DM) should be willing to pay to receive new information. In many cases, a DM seems to be receiving valuable information that is not about what seem to be the relevant events. This means that we cannot do a value of information calculation, at least not in the obvious way. For example, suppose that the DM is interested in learning a secret, which we assume for simplicity is a number between 1 and 1000. A priori, suppose that the DM takes each number to be equally likely, and so has probability .001. Learning the secret has utility, say, \$1,000,000; not learning it has utility 0. The number is locked in a safe, whose combination is a 40-digit binary numbers. What is the value to the DM of learning the first 20 digits of the combination? As far as value of information goes, it seems that the value is 0. The events relevant to the expected utility are the possible values of the secret; learning the combination does not change the probabilities of the numbers at all. This is true even if we put the possible combinations of the lock into the sample space. On the other hand, it is clear that people may well be willing to pay for learning the first 20 digits. It converts an infeasible problem (trying 2^{40} combinations by brute force) to a feasible problem (trying 2^{20} combinations). Because we charge for computation, it becomes straightforward to define a notion of value of computational information that captures the value of learning the first 20 digits.[6] The notion of value of (computational) conversation further extends these ideas by allowing a DM to interact with an informed observer before making a decision (as opposed to just getting some information). In making these definitions, we assumed, as we did in this paper, that all the relevant probabilities and utilities were given. It seems interesting to explore how things change in this setting if we charge for computing the probabilities and utilities.

- In our framework, the complexity profile depends only on the strategy profile of the players and the inputs to the machines. We could extend the framework to also require that players' beliefs be computable by Turing machines, and additionally charge for the complexity of those beliefs. For instance, a player might wish to play a strategy that can be justified by a "simple" belief: Consider a politician that needs to have a succinct explanation for his actions.

- We have focused here on Nash equilibrium. But, as we have seen, Nash equilibria do not always exist. This makes the question of what is the "right" solution concept for comutational games of particlar intereset. We could certainly consider other solution concepts, such as rationalizability, in the computational context. With rationalizability, we might want to take seriously the cost of computing the beliefs that rationalize the choice of a strategy.

- It would be interesting to provide epistemic characterization of various solution concepts for machine games. It seems relatively straightforward to define Kripke structures for machine games that model agents' beliefs about their own and others complexities and utilities, and their belief about other agents' beliefs about complexity and utility (as well as their beliefs about beliefs etc.). We do not yet know if we can provide epistemic characterizations using such Kripke structures.

- A natural next step would be to introduce notions of computation in the epistemic logic.

[6] Our notion of value of computational information is related to, but not quite the same as, the notion of *value of computation* introduced by Horvitz [1987, 2001]; see [Halpern and Pass 2010] for further discussion.

There has already been some work in this direction (see, for example, [Halpern, Moses, and Tuttle 1988; Halpern, Moses, and Vardi 1994; Moses 1988]). We believe that combining the ideas of this paper with those of the earlier papers will allow us to get, for example, a cleaner knowledge-theoretic account of zero knowledge than that given by Halpern, Moses, and Tuttle [1988]. A first step in this direction is taken in [Halpern, Pass, and Raman 2009].

- Finally, it would be interesting to use behavioral experiments to, for example, determine the "cost of computation" in various games (such as finitely repeated prisoner's dilemma).

8 Acknowledgments

The second author wishes to thank Silvio Micali and abhi shelat for many exciting discussions about cryptography and game theory, Ehud and Adam Kalai for enlightening discussions, and Dongcai Shen for useful comments on the paper. We also thank Tim Roughgarden for encouraging us to think of conditions that guarantee the existence of Nash equilibrium.

References

Abraham, I., D. Dolev, R. Gonen, and J. Y. Halpern (2006). Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *Proc. 25th ACM Symposium on Principles of Distributed Computing*, pp. 53–62.

Agrawal, M., N. Keyal, and N. Saxena (2004). Primes is in P. *Annals of Mathematics 160*, 781–793.

Aumann, R. (1985). What is game theory trying to accomplish. In K. Arrow and S. Honkapohja (Eds.), *Frontiers of Economics*, Volume 29, pp. 28–99. Oxford, U.K.: Basil Blackwell.

Aumann, R. J. (1987). Correlated equilibrium as an expression of Bayesian rationality. *Econometrica 55*, 1–18.

Axelrod, R. (1984). *The Evolution of Cooperation*. New York: Basic Books.

Ben-Sasson, E., A. Kalai, and E. Kalai (2007). An approach to bounded rationality. In *Advances in Neural Information Processing Systems 19 (Proc. of NIPS 2006)*, pp. 145–152.

Blum, M. and S. Micali (1984). How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing 13*(4), 850–864.

Conitzer, V. and T. Sandholm (2004). Computational criticisms of the revelation principle. In *Proc. Fifth ACM Conference on Electronic Commerce*, pp. 262–263. A longer version appears in *Proc. Workshop on Agent Mediated Electronic Commerce (AMEC V)*, 2003.

Dodis, Y., S. Halevi, and T. Rabin (2000). A cryptographic solution to a game theoretic problem. In *CRYPTO 2000: 20th International Cryptology Conference*, pp. 112–130. Springer-Verlag.

Fagin, R., J. Y. Halpern, Y. Moses, and M. Y. Vardi (1995). *Reasoning About Knowledge*. Cambridge, Mass.: MIT Press. A slightly revised paperback version was published in 2003.

Forges, F. (1986). An approach to communication equilibria. *Econometrica 54*(6), 1375–85.

Forges, F. (1990). Universal mechanisms. *Econometrica 58*(6), 1341–64.

Goldreich, O. (2001). *Foundations of Cryptography, Vol. 1*. Cambridge University Press.

Goldreich, O., S. Micali, and A. Wigderson (1986). Proofs that yield nothing but their validity and a methodology of cryptographic design. In *Proc. 27th IEEE Symposium on Foundations of Computer Science*.

Goldwasser, S., S. Micali, and C. Rackoff (1989). The knowledge complexity of interactive proof systems. *SIAM Journal on Computing 18*(1), 186–208.

Halpern, J. Y., Y. Moses, and M. R. Tuttle (1988). A knowledge-based analysis of zero knowledge. In *Proc. 20th ACM Symposium on Theory of Computing*, pp. 132–147.

Halpern, J. Y., Y. Moses, and M. Y. Vardi (1994). Algorithmic knowledge. In *Theoretical Aspects of Reasoning about Knowledge: Proc. Fifth Conference*, pp. 255–266.

Halpern, J. Y. and R. Pass (2009a). A computational game-theoretic framework for cryptography. unpublished manuscript.

Halpern, J. Y. and R. Pass (2009b). Sequential equilibrium and perfect equilibrium in games of imperfect recall. Unpublished manuscript.

Halpern, J. Y. and R. Pass (2010). I don't want to think about it now: Decision theory with costly computation. In *Principles of Knowledge Representation and Reasoning: Proc. Twelfth International Conference (KR '10)*.

Halpern, J. Y., R. Pass, and V. Raman (2009). An epistemic characterization of zero knowledge. In *Theoretical Aspects of Rationality and Knowledge: Proc. Twelfth Conference (TARK 2009)*, pp. 156–165.

Halpern, J. Y. and L. C. Rêgo (2006). Extensive games with possibly unaware players. In *Proc. Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 744–751. Full version available at arxiv.org/abs/0704.2014.

Håstad, J. (1989). Almost optimal lower bounds for small depth circuits. *Randomness and Computation, Advances in Computing Reasearch 5*, 143–170.

Heifetz, A., M. Meier, and B. Schipper (2007). Unawareness, beliefs and games. In *Theoretical Aspects of Rationality and Knowledge: Proc. Eleventh Conference (TARK 2007)*, pp. 183–192. Full paper available at www.econ.ucdavis.edu/faculty/schipper/unawprob.pdf.

Horvitz, E. (1987). Reasoning about beliefs and actions under computational resource constraints. In *Proc. Third Workshop on Uncertainty in Artificial Intelligence (UAI '87)*, pp. 429–444.

Horvitz, E. (2001). Principles and applications of continual computing. *Artificial Intelligence 126*, 159–196.

Kalai, E. (1990). Bounded rationality and strategic complexity in repeated games. In *Game Theory and Applications*, pp. 131–157. San Diego: Academic Press.

Kreps, D., P. Milgrom, J. Roberts, and R. Wilson (1982). Rational cooperation in finitely repeated prisoners' dilemma. *Journal of Economic Theory 27*(2), 245–252.

Kreps, D. M. and R. B. Wilson (1982). Sequential equilibria. *Econometrica 50*, 863–894.

Kuhn, H. W. (1953). Extensive games and the problem of information. In H. W. Kuhn and A. W. Tucker (Eds.), *Contributions to the Theory of Games II*, pp. 193–216. Princeton, N.J.: Princeton University Press.

Lipton, R. J. and E. Markakis (2004). Nash equilibria via polynomial equations. In *Proc. LATIN 2004: Theoretical Informatics*, pp. 413–422.

Lipton, R. J., E. Markakis, and A. Mehta (2003). Playing large games using simple strategies. In *Proceedings of the 4th ACM Conference on Electronic Commerce*, pp. 36–41.

Lord, C., L. Ross, and M. Lepper (1979). Belief assimilation and attitude polarization: the effects of prior theories on subsequently considered evidence. *Journal of Personality and Social Psychology 37*(11), 2098–2109.

Megiddo, N. (1989). On computable beliefs of rational machines. *Games and Economic Behavior 1*, 144–169.

Megiddo, N. and A. Wigderson (1986). On play by means of computing machines. In *Theoretical Aspects of Reasoning about Knowledge: Proc. 1986 Conference*, pp. 259–274.

Moses, Y. (1988). Resource-bounded knowledge. In *Proc. Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pp. 261–276.

Mullainathan, S. (2002). A memory-based model of bounded rationality. *Quarterly Journal of Economics 117*(3), 735–774.

Myerson, R. (1979). Incentive-compatibility and the bargaining problem. *Econometrica 47*, 61–73.

Nash, J. (1951). Non-cooperative games. *Annals of Mathematics 54*, 286–295.

Neyman, A. (1985). Bounded complexity justifies cooperation in finitely repated prisoner's dilemma. *Economic Letters 19*, 227–229.

Papadimitriou, C. H. and T. Roughgarden (2005). Computing equilibria in multi-player games. In *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms*, pp. 82–91.

Papadimitriou, C. H. and M. Yannakakis (1994). On complexity as bounded rationality. In *Proc. 26th ACM Symposium on Theory of Computing*, pp. 726–733.

Piccione, M. and A. Rubinstein (1997). On the interpretation of decision problems with imperfect recall. *Games and Economic Behavior 20*(1), 3–24.

Rabin, M. (1998). Psychology and economics. *Journal of Economic Literature XXXVI*, 11–46.

Rubinstein, A. (1986). Finite automata play the repeated prisoner's dilemma. *Journal of Economic Theory 39*, 83–96.

Samuelson, W. and R. Zeckhauser (1998). Status quo bias in decision making. *Journal of Risk and Uncertainty 1*, 7–59.

Simon, H. A. (1955). A behavioral model of rational choice. *Quarterly Journal of Economics 49*, 99–118.

Tarski, A. (1951). *A Decision Method for Elementary Algebra and Geometry* (2nd ed.). Univ. of California Press.

Turing, A. M. (1937). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2 42*, 230–265.

Tversky, A. and D. Kahneman (1981). The framing of decisions and the psychology of choice. *Science 211*, 453–58.

Urbano, A. and J. E. Vila (2004). Computationally restricted unmediated talk under incomplete information. *Economic Theory 23*(2), 283–320.

Walker, G. and D. Walker (2004). *The Official Rock Paper Scissors Strategy Guide*. New York: Simon & Schuster, Inc.

Walker, M. and J. Wooders (2001). Minimax play at wimbledon. *American Economic Review 91*(5), 1521–1538.

Wilson, A. (2002). Bounded memory and biases in information processing. Manuscript.

Yao, A. (1982). Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science*, pp. 80–91.