Probabilistic Oracle Machines and Nash Equilibria (Draft)

Jessica Taylor – jessica@intelligence.org
Benja Fallenstein – benja@intelligence.org

February 7, 2015

1 Introduction

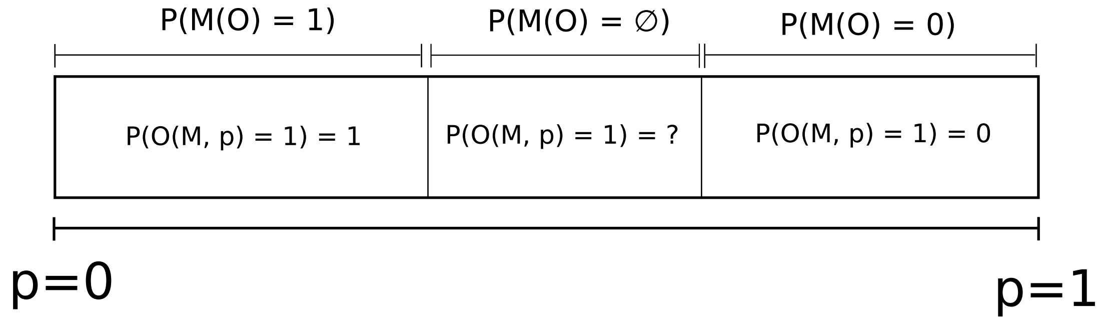
In many situations, programs would like to predict the output of other programs. They could simulate the other program in order to do this. However, this method fails when there are any cycles (i.e. program A is concerned with the output of program B which is concerned with the output of program A). Furthermore, if a procedure to determine the output of another program existed, then it would be possible to construct a liar's paradox of the form "if I return 1, then return 0, otherwise return 1".

These paradoxes could be resolved by using probabilities. Define $\mathcal{M}$ to be the set of probabilistic Turing machines (which will here be defined as Turing machines that are allowed to flip coins that have rational probabilities of coming up heads) that can call an oracle O during their execution, and may return either 0 or 1. O functions as a (possibly randomized) procedure that predicts whether another program returns 1 with at least some probability. O must always return either 0 or 1. Additionally, we would like O to satisfy a reflection principle: for each $M \in \mathcal{M}$ and probability p, we would like

- $P(M(O) = 1) > p \Rightarrow P(O(M, p) = 1) = 1$
- $P(M(O) = 0) > 1 - p \Rightarrow P(O(M, p) = 0) = 1$

where $M(O)$ is a random variable that results from observing the return value of program M with access to oracle O. Note that, although we focus on always-halting machines in this paper, in general $M(O)$ might never halt, in which case we take $M(O) = \emptyset$, a special symbol distinct from 0 and 1. The reflection principle roughly states that the oracle correctly predicts the 1-returning probabilities of machines when those machines have access to this same oracle.

Graphically, we could display this requirement as follows:



What happens if we define a liar's paradox that tries to output the opposite of what it outputs? This machine can be defined using quining as $M(O) = 1 - O(M, 0.5)$. But we can set $P(O(M, 0.5) = 1) = P(O(M, 0.5) = 0) = 0.5$, and there is no paradox: the program outputs 1 half the time and 0 the other half of the time. In fact, there does exist an oracle satisfying the reflection principle. We will prove a weaker version of this result later.

2 Recovering Nash equilibria from expected utility maximization

As one application of these machines, suppose we have a collection of m players that are playing a normal form game with each other. Each player i may select any pure strategy in the set of actions A_i. After all strategies $a_1, ..., a_n$ have been selected, each player i receives utility $U_i(a_1, ..., a_n)$. W will write $U_i(a_{-i}, x)$ as shorthand for $U_i(a_1, ..., a_{i-1}, x, a_{i+1}, ..., a_n)$. For now, assume each player has 2 actions, so $A_i = \{0, 1\}$; we will see later how to extend this framework to more than 2 actions.

We want to specify each player as a probabilistic Turing machine having access to an oracle, and we want this machine to always halt within a bounded number of steps. Each player chooses an action that maximizes expected utility. Therefore, each player will use the oracle to predict other players' strategies and then select a utility-maximizing action.

Assume without loss of generality that all utilities are between 0 and 1. With this assumption, we can go on to construct a machine E_i whose probability of returning 1 is equal to $\mathbb{E}\left[\frac{U_i(a_{-i},1)-U_i(a_{-i},0)+1}{2}\right]$. If this machine returns 1 with probability at least 0.5, then $U_i(a_{-i}, 1) \geq U_i(a_{-i}, 0)$, and therefore 1 is a utility-maximizing action for player i; conversely, if this machine returns 0 with probability at least 0.5, then 0 is a utility-maximizing action for player 1. Therefore, we can represent player 1 as a machine $M_i = O(E_i, 1/2)$.

E_i will simulate each other player (which will just end up making a similar oracle call) to get a_{-i}, and then flip a coin that has probability $\frac{U_i(a_{-i},1)-U_i(a_{-i},0)+1}{2}$ of returning 1. Since the return value of E_i is an unbiased estimate of $\mathbb{E}\left[\frac{U_i(a_{-i},1)-U_i(a_{-i},0)+1}{2}\right]$, the probability of returning 1 will be this expectation. This does require some quining in order to define this machine as having access to the source code of other machines that have access to this machine's source code; however, Kleene's recursion theorem ensures that we can do this.

Now, we note that if O satisfies the reflection principle, then the mixed strategy specified by the probabilities O assigns to $O(E_1, 1/2), ..., O(E_n, 1/2)$ form a Nash equilibrium. This is because each player plays a best response to the other players' mixed strategies as specified by the oracle.

How can we extend this analysis to players with more than 2 strategies? Suppose player i has available strategies $\{0, 1, ..., k_i - 1\}$ and is currently picking between picking 0 and picking a different strategy. We would like to construct a machine whose probability of returning 1 is equal to $\mathbb{E}\left[\frac{U_i(a_{-i}, a_{i,\geq 1}) - U_i(a_{-i}, 0) + 1}{2}\right]$, where $a_{i,\geq 1}$ is a random variable representing what player i would do if strategy 0 were unavailable. Naturally, we can represent this random variable using more machines: one choosing between strategy 1 and any strategies after 1, one choosing between strategy 2 and any strategies after 2, and so on. Recursively, define machines $E_{i,\geq j}$ and random variables $a_{i,\geq j}$ such that:

$$\begin{aligned}
P(E_{i,\geq j} = 1) &= \frac{U_i(a_{-i}, a_{i,\geq j+1}) - U_i(a_{-i}, j) + 1}{2} && \text{for } j \leq k_i - 1 \\
a_{i,\geq k_i - 1} &= k_i - 1 \\
a_{i,\geq j} &= a_{i,\geq j+1} && \text{for } j \leq k_i - 2, O(E_{i,\geq j}, 1/2) = 1 \\
a_{i,\geq j} &= j && \text{for } j \leq k_i - 2, O(E_{i,\geq j}, 1/2) = 0 \\
a_i &= a_{i,\geq 0}
\end{aligned}$$

What is interesting about this construction is that the players ended up playing a Nash equilibrium by predicting the output of each other's programs. Unlike in standard game theory, other players are not primitive entities: they are simply programs that happen to be constructed in a particular way.

3 Equivilance with Nash equilibria

So, we can use the probabilistic oracle to find Nash equilibria in arbitrary normal-form games. It is interesting that we can also go the other direction: given any finite set of probabilistic Turing machines with access to an oracle that only call other programs in this set, and which each always halts within time t, we can construct a normal-form game whose Nash equilibria correspond to reflective oracles assigning probabilities to all these machines and all queries they might make. Note that it is easy to ensure that we have a finite set of machines that only call other machines in this set by bounding the length and computing power of each machine: due to bounded computing power, it will be impossible to construct a query about a large program to pass to the oracle.

To begin, we must start by showing that we can construct normal form games whose Nash equilibria must satisfy some of an important class of polynomial constraints on players' mixed strategies $p_1, ..., p_n$. These polynomial constraints will be used to encode the reflection principle.

Suppose we have an incomplete specification of a normal form game in which each player has 2 possible actions 0 and 1, and we would like to modify it so that in any Nash equilibrium, player i will have a specific probability p of playing 1. We do this by adding a new player j to the game and setting U_i and U_j such that they play a variant of the matching pennies game:

	$a_j = 0$	$a_j = 1$
$a_i = 0$	$U_i = 1, U_j = 0$	$U_i = 0, U_j = p$
$a_i = 1$	$U_i = 0, U_j = 1 - p$	$U_i = 1, U_j = 0$

In this game, in every Nash equilibrium, we have $p_i = p$. This claim is proven in the appendix.

What if we want the probability p to depend on the mixed strategies of other players $k_1, ..., k_m$? We could choose to vary p as a function of some of the other players' actions $p = f(a_{k_1}, ..., a_{k_m})$ for any f whose image is a subset of $[0, 1]$. Due to independence among all actions and maximization of expected utility, we can treat the effective value of p (and therefore p_i) as the expected value of p across independent draws from the mixed strategies:

$$\begin{aligned}
p_i &= \mathbb{E}_{a_{k_1} \sim \text{Bernoulli}(p_{k_1}), ..., a_{k_m} \sim \text{Bernoulli}(p_{k_m})}[f(a_{k_1}, ..., a_{k_m})] \\
&= \sum_{a_{k_1} \in \{0,1\}, ..., a_{k_m} \in \{0,1\}} f(a_{k_1}, ..., a_{k_m}) \prod_{l=1}^{m} P_{a_{k_l} \sim \text{Bernoulli}(p_{k_l})}(a_{k_l}) \\
&= \sum_{a_{k_1} \in \{0,1\}, ..., a_{k_m} \in \{0,1\}} f(a_{k_1}, ..., a_{k_m}) \prod_{l=1}^{m} (a_{k_l} p_{k_l} + (1 - a_{k_l})(1 - p_{k_l}))
\end{aligned}$$

So we can require that in a Nash equilibrium, p_i is a polynomial function of $p_{k_1}, ..., p_{k_m}$ with this form.

Now let us see how to encode the reflection principle as a set of polynomial constraints of this form. To simplify matters, let us note that any finite probabilistic oracle machine can be expanded into a finite state machine where each state is one of the following kinds:

- A state that immediately halts the program and returns either 0 or 1.
- A state that, for some rational number p, goes to some state with probability p and another state with probability $1 - p$.
- A state that queries the oracle with some (M, p), goes to some state if $O(M, p) = 1$, and goes to a different state if the oracle returns 0.

Furthermore, note that it is unnecessary to have both the notion of a machine and a program state; we could represent each program state as a machine that simulates the original machine starting from the given state.

Assume we can enumerate all possible machine states $s_1, ..., s_m$ and oracle calls of the form (s_i, q). Now we will want to create a normal form game that has players representing each state and oracle call. We can map each machine state s_i to a player with index i, and we can map each oracle call (s_i, q) to a player with index $g(s_i, q) > m$. We would like these players' mixed strategies to be set so that for any s_i, p_i equals the probability that s_i eventually returns 1, and we would like $p_{g(s_i,q)}$ to equal the probability that the oracle call (s_i, q) returns 1. Consider the different types of machine states:

- For a state s_i that immediately halts and returns 0, it is trivial to set player i's utility function such that they always play 0 and so $p_i = 0$. Same goes for a state that returns 1.

- For a state that goes into state s_l with rational probability q and state s_r with probability $1-p$, we would like $p_i = qp_l + (1-q)p_r$. Luckily this is a polynomial of p_l and p_r that has the form we want.

- For a state that queries the oracle with (s_o, q) and then goes into state s_l if the oracle returns 0 and s_r if it returns 1, we would like $p_i = p_{g(o,q)}p_l + (1 - p_{g(o,q)})p_r$. This is also a polynomial having the form we want.

We also need to directly encode the reflection principle. For this, we set player $g(i,p)$'s utility function to the following:

	$a_i = 0$	$a_i = 1$
$a_{g(i,p)} = 0$	$U_{g(i,p)=p}$	$U_{g(i,p)=p}$
$a_{g(i,p)} = 1$	$U_{g(i,p)} = 0$	$U_{g(i,p)} = 1$

Due to this utility function, in any Nash equilibrium we have $p > p_i \Rightarrow p_{g(i,p)} = 0$ and $p < p_i \Rightarrow p_{g(i,p)} = 1$, which encodes the reflection principle.

So, we can set up a game such that we can compute, from any Nash equilibrium, a reflective oracle for this set of machines. Therefore, we have shown that the problem of computing a reflective oracle defined on a finite set of programs all terminating within some time can be reduced to the problem of computing a Nash equilibrium. This shows by construction that such a reflective oracle always exists.

4 Appendix: Nash Equilibria in Matching Pennies

We have a 2 by 2 game: Player Row has strategies Up and Down, player Column has strategies Left and Right. The payoffs of (Row, Column) are as follows:

$(1,0)$	$(0,x)$
$(0,1-x)$	$(1,0)$

Write p for the probability that Row plays Down. I claim that for every $x \in [0,1]$, at Nash equilibrium we have $p = x$. (Since this is a good old finite game, we know that a Nash equilibrium always exists.)

- Case 1: $0 < x < 1$.

  Suppose that there's a Nash equilibrium where Column plays Left. Then Row would play Up, but then Column would strictly prefer Right, contradiction.

  Suppose that there's a Nash equilibrium where Column plays Right. Then Row would play Down, but then Column would strictly prefer Left, contradiction.

  So at every Nash equilibrium, Column must mix between strategies.

  Thus, at equilibrium, Column must be indifferent between Left and Right.

  This is equivalent to $p(1-x) = (1-p)x$.

  This implies $p > 0$, since otherwise we'd have $0(1-x) = (1-0)x$, i.e. $0 = x$, but we assumed $0 < x < 1$.

Thus, we can divide the equation by px, yielding:

$$\begin{aligned} & (1-x)/x = (1-p)/p \\ \Leftrightarrow & 1/x - 1 = 1/p - 1 \\ \Leftrightarrow * & 1/x = 1/p \\ \Leftrightarrow & x = p \end{aligned}$$

- Case 2: $x = 0$.

  This gives us the following payoff matrix:

$(1,0)$	$(0,0)$
$(0,1)$	$(1,0)$

  Suppose that there's a Nash equilibrium with $p > 0$. Then at this equilibrium, Column must play Left; but if Column plays Left, then Row strictly prefers Up, contradiction to $p > 0$. Hence, we must have $p = 0 = x$.

- Case 3: $x = 1$.

  This gives us the following payoff matrix:

$(1,0)$	$(0,1)$
$(0,0)$	$(1,0)$

  Suppose that there's a Nash equilibrium with $p < 1$. Then at this equilibrium, Column must play Right; but if Column plays Right, then Row strictly prefers Down, contradiction to $p < 1$. Hence, we must have $p = 1 = x$.