RECURSIVELY ENUMERABLE LANGUAGES HAVE FINITE STATE INTERACTIVE PROOFS

Richard J. Lipton

CS-TR-213-89

March 1989

Recursively Enumerable Languages have Finite State Interactive Proofs

Richard J. Lipton*
Computer Science Department
Princeton University

Abstract

We show that interactive proof systems with 2-way probabilistic finite state verifiers can accept any recursively enumerable language. The same proof method also shows that the emptiness problem for 1-way probabilistic finite state machines is undecidable.

1 Introduction

We study the class of *Interactive Proof Systems* (IPS) introduced by Goldwasser, Micali, and Rackoff [6] and Babai [1]. Roughly, an IPS for a language L is a 2-party protocol between a "prover" and a "verifier". The more powerful prover must convince the verifier that elements in L are actually in L. We study the case where the prover is unrestricted and the verifier is a probabilistic finite state machine with the ability to flip "private" coins. Dwork and Stockmeyer [4] and the work of Condon [2], show that such IPS's are very powerful: an IPS with a probabilistic finite state machine as a verifier can accept all languages in E, the class of exponential-time computable languages. In their notation, $E \subseteq IP(2pfa)$. Here $IP(2pfa)$ is the class of languages L such that for each $\epsilon > 0$ there is a 2-party protocol between an unrestricted prover and a finite state probabilistic verifier. The verifier has a 2-way input tape with the input x. Then the following must be true:

1. for all x in L, for some prover, the IPS accepts the input with probability at least $1 - \epsilon$.
2. for all x not in L, for all provers, the IPS rejects with probability at least $1 - \epsilon$.

We are able to show that IPS's are very powerful. In particular, we can show that the following theorem is true:

Theorem 1.1 *For any recursively enumerable language L, L is in $IP_w(2pfa)$.*

Again in the notation of [4], this is a weaker class of IPS's. The key is that we do not insist that these systems always halt. Thus, condition (2) in the above definition is changed to: for all x not in L, for all provers, the IPS accepts with probability at most ϵ. Dwork and Stockmeyer [4] show that pushdown automata with "private" coins can accept any recursively enumerable language. Condon [3] has observed that "almost always halting" is a strong restriction:

*Work supported by DARPA and ONR contracts N00014-85-C-0456 and N00014-85-K-0465, and by NSF Cooperative Agreement DCR-8420948

Theorem 1.2 *If L is in $IP(2pfa)$, then L is a recursive language.*

The proof of this last remark is simple: since the system halts all but ϵ of the time, a brute-force search of all the computations must eventually be able to halt. That is search until "most" of the computations have either accepted or rejected.

The proof of theorem 1.1 requires us to extend a result of Frievalds [5]. He shows that the emptiness problem for 2-way probabilistic finite state machines is undecidable. We can use our method to prove:

Theorem 1.3 *The emptiness problem for 1-way probabilistic finite state machines is undecidable.*

The emptiness problem appears to have first been raised in the late 1960's. See, for example, Paz [7]. The key insight into this generalization is that it is possible to "simulate" the 2-way input tape by a 1-way input tape. The 1-way tape repeats over and over the contents of the 2-way tape. The critical point is that the 1-way tape can "lie", i.e. it need not repeat exactly the same string each time. Thus, the 1-way machine must be a kind of verifier and check that no cheating occurs. In this way ideas from IPS's play an essential role in the proof of theorem 1.3. Since we often measure the importance of a new area by how well it solves existing problems, theorem 1.3 is evidence for the importance of IPS's.

A further comment on our main theorem 1.1 is that our proof does not require real interaction between the prover and the verifier. The prover sends a "proof" to the verifier for checking, but the prover never needs any information from the verifier. Thus, we can show that even this restricted class of IPS's is enough to accept all recursively enumerable languages.

2 Proof of Theorem 1.1

In this section we will prove the main theorem that shows how any recursively enumerable language can be accepted by some IPS with only a probabilistic finite state machine as a verifier. We first need a technical lemma about conditional probability.

Lemma 2.1 *Let A,B be independent events with*

$$0 < kPr[B] = Pr[A] < 1.$$

where $k \geq 1$. Then,

$$Pr[A|A\bar{B} \vee \bar{A}B] \geq 1 - \frac{1}{k+1}.$$

Proof: Let $\alpha = Pr[A]$ and $\beta = Pr[B]$. Then, $0 < k\beta = \alpha < 1$. Now, since A and B are independent events,

$$\begin{aligned} Pr[A|A\bar{B} \vee \bar{A}B] &= \frac{Pr[A\bar{B}]}{Pr[A\bar{B}] \vee Pr[\bar{A}B]} \\ &= \frac{\alpha(1-\beta)}{\alpha(1-\beta)+(1-\alpha)\beta} \\ &= \frac{1}{1+\delta} \end{aligned}$$

where δ is

$$\frac{(1-\alpha)\beta}{(1-\beta)\alpha}.$$

Since $\alpha = k\beta$ and $k \geq 1$ it follows that

$$\begin{aligned} 1+\delta &= 1+\frac{(1-k\beta)}{(1-\beta)k} \\ &\leq 1+\frac{1}{k} \\ \frac{1}{1+\delta} &\geq \frac{1}{1+1/k} \\ &\geq 1-\frac{1}{k+1}. \end{aligned}$$

Therefore, $Pr[A|A\bar{B} \vee \bar{A}B] \geq 1 - \frac{1}{k+1}$ as required. □

We next consider a kind of *game* that our finite state verifiers will use to check the prover. We assume that the prover sends a string of the following form to the verifier: $a^i b^j$. We further assume that the finite state verifier receives each character one at a time left-to-right. We also fix a large number p. The verifier will perform five independent computations:

1. For each letter a, the verifier flips two fair coins.
2. For each letter b, the verifier flips two fair coins.
3. For each letter a and each letter b, the verifier flips one fair coin.
4. For each letter a and each letter b, the verifier flips one fair coin.
5. The verifier checks that $i \equiv j \bmod p$ is true.

Note, these are all independent computations. The outcome of the game is determined as follows: if step (5) discovers that i and j are not equal modulo p, then the game outcome is *not-equal.* Therefore, assume that this is not the case. Let A be true if either step (1) or step (2) gets only heads; let B be true if either step (3) or step (4) gets only heads. Then say that the outcome of the game is a *tie* if both A and B are false or both are true. Say that the outcome is *double wins* if A is true and B is false; say that the outcome is *sum wins* if B is true and A is false. Say that the outcome is a *win* provided either double or sum wins.

Lemma 2.2 *For any fixed p, the outcome of a game can be computed by a probabilistic finite state machine.*

Proof: Clearly, each of the five subcomputations can be done with a finite number of states. Each computation need only remember one bit about the sequence of coin flips, i.e. were all the coins heads. Since the decision on what the game's outcome is clearly finite state, the claim follows. □

The critical lemma is the following:

Lemma 2.3 *Consider the action of the game on the string $a^i b^j$ with number p and $i \equiv j \bmod p$.*

1. *If $i = j$, then $Pr[double\ wins|win] = Pr[sum\ wins|win]$.*
2. *If $i \neq j$, then $Pr[double\ wins|win] \geq 1 - \frac{1}{2^{p-1}+1}$.*
3. *If $i \neq j$, then $Pr[sum\ wins|win] \leq \frac{1}{2^{p-1}+1}$.*

Proof: First, suppose that $i = j$. Then each of the computations flips exactly the same number of independent coins, i.e. $2i = 2j = i + j$. Thus, by symmetry it follows that (1) is true. Now assume that $i < j$. Now let A denote the event that double wins; also let B denote the event that sum wins. Then,

$$Pr[A] \geq Pr[2i \text{ heads in a row}] = 1/2^{2i}.$$

Also,

$$Pr[B] \leq 2Pr[i + j \text{ heads in a row}] = 1/2^{i+j-1}.$$

Thus, $Pr[A] \geq kPr[B]$ where k is 2^{j-i-1}. Since i is congruent to j modulo p, $j - i \geq p$. So $k \geq 2^{p-1}$. Lemma 2.1 then shows that $Pr[A|A\bar{B} \vee \bar{A}B]$ is at least $1 - \frac{1}{2^{p-1}+1}$. Now assume that $i > j$. The argument follows in exactly the same manner replacing i by j. This proves (2). Finally, (3) follows directly from (2) since either double or sum must win each game that has a winner. □

It is convenient to have one more definition concerning games. Say that a game is *fair* if the string is $a^i b^j$ where $i = j$; say it is *unfair* otherwise. Thus, lemma 2.3 states: in any fair game, double or sum are equally likely to win; in any unfair game, double is much more likely to win than sum. This game is based on the method used by Frievalds in [5]. The critical difference is that in his game one flips i coins and one flips j coins. If $i = j$, then both are equally likely to win. If $i \neq j$, then one is much more likely to win. However, he cannot predict who is going to be more likely to win, i or j. This is a key problem that our more complex game avoids. We know, if the game is unfair, *which* outcome is more likely, not just that *some* outcome is much more likely.

Let L be a recursively enumerable language. Then there is a *counter machine* that accepts exactly L. Our counter machines are standard; they have two counters and a read-only two-way input tape. At each step the machine operates as follows: it reads the contents of the input tape and tests whether or not each of its counters is zero or not. Then based on its current control state it may move the read-only tape left or right, increment a counter, decrement a counter provided it is nonzero, and finally enter its next state. As usual the counter machine starts in a fixed state, and it accepts by entering a fixed final state. It is well known that such machines are powerful enough to accept any recursively enumerable language. We encode the instantaneous descriptions (ID's) of such a machine as follows: $\#uc^k d^l \#$ here u is a symbol that encodes the control state, k is the value of the first counter, l is the value of the second counter.

We now are ready to prove theorem 1.1.

Proof: Let L be any recursively enumerable language and let $\epsilon > 0$ be selected. Fix a counter machine that accepts this language. We will show that there is a probabilistic finite state verifier that accepts exactly this language, i.e.,

1. If x is in L, then the verifier will accept with probability at least $1 - \epsilon$.
2. If x is not in L, then the verifier will accept with probability at most ϵ.

The plan is that the prover will repeatedly send to the verifier the sequence of ID's that correspond to the accepting computation. The verifier will check the ID's for correctness. More precisely, suppose that x is in the language L. Then, the prover will send the accepting computation to the verifier:

$$\#u_1 c^{k_1} d^{l_1} \#, \ldots, \#u_t c^{k_t} d^{l_t} \#.$$

We will call this string the "computation" and denote it by $\mathcal{C}$. Without loss of generality we can assume that t is even since the counter machine can be modified so it only accepts after an even number of steps. The prover will send this string repeatedly to the verifier. (Of course the prover can "cheat" and send

illegal strings or different ones each time.) The verifier will then check the computation $\mathcal{C}$ as follows. First, the verifier will check that it is "locally" correct. This includes: (i) checking that the initial and final states are correct; (ii) checking that the correct state transitions are taken. The latter requires that the verifier use its input read-only to simulate the input tape of the counter machine. That is the verifier moves the input tape according to the motions encoded in each ID. It constantly checks to see whether or not the ID's claim to read the wrong symbol from the input tape. It also requires that the verifier check that all zero tests are correct. Clearly, all these checks can be done with a deterministic finite state control. If any of these fail, then the verifier rejects.

The second, is to check that the computation $\mathcal{C}$ is "globally" correct. This means that each of the two counters is modelled correctly. Thus, it must be the case that, for each $i = 1, \ldots, t-1$,

$$k_{i+1} = k_i + r_i \quad (1)$$

$$l_{i+1} = l_i + s_i \quad (2)$$

where r_i (resp. s_i) is the action performed on the first (resp. second) counter, i.e. r_i (resp. s_i) is $-1, 0, 1$ provided the first (resp. second) counter is decremented, left alone, or incremented at the i^{th} step of the computation. Clearly, these global conditions are *not* checkable by a deterministic finite state verifier. Thus, we must use the probabilistic nature of the verifier to check them.

Fix a large number p and another large constant q. It is convenient to select their exact values later on. The verifier will have two "counters" D and S: each is initially 0 and can take values from 0 to q.

Now the verifier performs four computations as it receives the computation $\mathcal{C}$ from the prover. Let us denote these by A_i and B_i where i is 0 or 1. We will first describe A_0. This computation performs a finite state transduction on the computation $\mathcal{C}$ mapping it to

$$a^{k_2+r_2}b^{k_3}, a^{k_4+r_4}b^{k_5}, \ldots, a^{k_{2m-2}+r_{2m-2}}b^{k_{2m-1}}$$

where $m = t/2$. The key point is that r_i is easy to compute given the ID's. Thus, this transduction is computable with only a finite state control. Now the verifier will consider each of the pairs $a^{k_{2i}+r_{2i}}b^{k_{2i+1}}$ as a game. It will play these games in the order as they appear. The critical point is that the verifier needs this string only one bit at a time to play all the games. If some game's outcome is not-equal, then it will immediately reject. Otherwise, it will return the following information: whether or not *all* the games were won by double or whether or not *all* were won by sum. Clearly, it can do this with it's finite state control.

Essentially, A_0 is checking (1) for even i's. In the same way A_1 checks (1) for odd i's, B_0 checks (2) for even i's, and B_1 checks (2) for odd i's. Thus, A_1, B_0, and B_1 map the computation $\mathcal{C}$ to the following strings:

$$a^{k_1+r_1}b^{k_2}, a^{k_3+r_3}b^{k_4}, \ldots, a^{k_{2m-1}+r_{2m-1}}b^{k_{2m}}$$

and

$$a^{l_2+s_2}b^{l_3}, a^{l_4+s_4}b^{l_5}, \ldots, a^{l_{2m-2}+s_{2m-2}}b^{l_{2m-1}}$$

and

$$a^{l_1+s_1}b^{l_2}, a^{l_3+s_3}b^{l_4}, \ldots, a^{l_{2m-1}+s_{2m-1}}b^{l_{2m}}$$

They play the games exactly as A_0 does. If any game's outcome is not-equal, they immediately reject; otherwise, they return whether or not the games were all won by double or by sum.

Now the verifier updates its two counters D and S as follows. If double (resp. sum) won *all* the games played by A_0, A_1, B_0, B_1, then increase D (resp. S) by 1. The verifier then considers the values of the counters D and S:

1. If D and S are both less than q, then it gets another computation $\mathcal{C}$ from the prover and repeats the four computations A_0, A_1, B_0, B_1 as before.
2. If D equals q, then accept if S is non-zero and reject if it is 0.
3. If S equals q, then accept.

It remains to show that this protocol is correct. There are two cases. First, suppose that the input x is in the language L. Then there is an accepting computation; hence, we can assume that the prover always sends this accepting computation to the verifier. Since the computation is correct every game played by the A_0, A_1, B_0, B_1 is fair. Thus, the probability that all are won by double is exactly the same as the probability that all are won by sum. Now the only way that the verifier can reject is for D to equal q while S is still 0. But, if q is large enough the probability of this can be made very small, i.e. for q large enough, it is much smaller than ϵ. (Note, q must be large but it does not depend on p.) Therefore, in this case the verifier accepts with probability at least $1 - \epsilon$.

Second, assume that x is not in the language L. Then, there is no accepting computation and each "computation" $\mathcal{C}$ that the prover sends to the verifier must be incorrect. Clearly, if the prover ever sends a computation $\mathcal{C}$ that violates any condition except (1) and (2), the verifier will always discover it. Thus, we can assume that only (1) or (2) is violated. We can also assume that no game played ever ends in the outcome, not-equal. Otherwise, the verifier will immediately reject. The key insight is that for each computation $\mathcal{C}$, at least one game played by A_0, A_1, B_0, B_1 on must be unfair. Therefore, the probability that double wins all these games is always much higher than the probability that sum wins all these games: this follows directly from lemma 2.3. Now assume that the verifier eventually makes a decision to accept or reject. For it to accept, S must be non-zero before D is larger than q. But, if p is large enough compared with q, the probability of this happening is much smaller than ϵ. This completes the proof of the theorem. □

3 Proof of Theorem 1.3

The emptiness problem for a 1-way probabilistic finite state machine is the problem of determining whether or not there is some input that the machine accepts with probability at least $1 - \epsilon$.

We will now prove theorem 1.3.

Proof: Suppose that the emptiness problem is decidable. We will show that the halting problem is decidable for counter machines with no input. Since this is impossible, this will complete the proof. Let M be a counter machine. We can construct a verifier as in the proof of theorem 1.1. Since the prover constructed in this proof never gets information from the verifier, we can consider the 1-way input tape as the "prover". If the verifier needs the next bit from the prover, it just reads the next bit from the tape. As before if there is an accepting computation, then the verifier accepts with probability at least $1 - \epsilon$. If on the other hand, there is no accepting computation, then the verifier can only be "fooled" into accepting with probability less than ϵ. □

Acknowledgement

I would like to thank Anne Condon for asking me how powerful finite state IPS are. I would also like to thank Joan Feigenbaum for many helpful comments.

References

[1] L. Babai, *Trading Group Theory for Randomness*, Proc. 17th STOC, 1985, pp. 421-429.

[2] A. Condon, *Computational Models of Games*, Ph.D. Thesis, Tech. Report 87-04-04, Computer Science Dept., University of Washington, Seattle, WA, 1987.

[3] A. Condon, private communication, Dec. 1988.

[4] C. Dwork, L. Stockmeyer, *Interactive Proof Systems with Finite State Verifiers*, Tech. Report RJ 6262, IBM Research Division, Almaden Research Center, San Jose, CA, 1988.

[5] R. Frievalds, *Probabilistic Two-way Machines*, Proc. International Symposium on Mathematical Foundations of Computer Science, vol. 118, Springer-Verlag, New York, 1981, pp.33-45.

[6] S. Goldwasser, S. Micali, and C. Rackoff, *The Knowledge Complexity of Interactive Proofs*, Proc. of 17th STOC, 1985, ppp. 291-304.

[7] A. Paz, *Introduction to Probabilistic Automata*, Academic Press, 1971.

720