Finite identification of functions by teams with success ratio $\frac{1}{2}$ and above

Sanjay Jain
Department of Information Systems and Computer Science
National University of Singapore
Lower Kent Ridge Road, Singapore 0511
Republic of Singapore
Email: sanjay@iscs.nus.sg

Arun Sharma
School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia
Email: arun@cse.unsw.edu.au

Mahendran Velauthapillai
Department of Computer Science
Georgetown University
Washington, D. C. 20057, USA
Email: mahe@cs.georgetown.edu

March 12, 2007

Abstract

Consider a scenario in which an algorithmic machine, **M**, is being fed the graph of a computable function f. **M** is said to *finitely identify* f just in case after inspecting a finite portion of the graph of f it emits its first conjecture which is a program for f, and it never abandons this conjecture thereafter. A *team* of machines is a multiset of such machines. A team is said to be successful just in case each member of some nonempty subset, of predetermined size, of the team is successful. The ratio of the number of machines required to be successful to the size of the team is referred to as the *success ratio* of the team. The present paper investigates the finite identification of computable functions by teams of learning machines. The results presented complete the picture for teams with success ratio $\frac{1}{2}$ and greater.

It is shown that at success ratio $\frac{1}{2}$, introducing redundancy in the team can result in increased learning power. In particular it is established that larger collections of functions can be learned by employing teams of 4 machines and requiring at least 2 to be successful than by employing teams of 2 machines and requiring at least 1 to be successful. Surprisingly, it is also shown that introducing further redundancy at success ratio $\frac{1}{2}$ does not yield any extra learning power. In particular, it is shown that the collections of functions that can be finitely identified by a team of $2m$ machines requiring at least m to be successful is the same as:

- the collections of functions that can be finitely identified by a team of 4 machines requiring at least 2 to be successful, if m is even, and
- the collections of functions that can be identified by a team of 2 machines requiring at least 1 to be successful, if m is odd.

These latter results require development of sophisticated simulation techniques.

1 Introduction

Consider a typical learning situation involving a learner attempting to learn a concept. The learner is presented with data about the concept, and from time to time as data is being received, the learner conjectures a sequence of hypotheses. Successful learning is said to take place, just in case, the learner eventually conjectures a hypothesis that correctly explains the concept and which hypothesis the learner never abandons. This criterion of success is essentially the notion of *learning in the limit* formalized by Gold [Gol67] in his seminal paradigm of *identification*.

A team of learners is a multiset of learners. A team is said to be successful just in case each member of some nonempty subset of the team is successful. The ratio of the number of learners required to be successful to the size of the team is referred to as the *success ratio* of the team. The problem of learning in the limit a program for a recursive function from its graph by a team of *deterministic* machines was motivated by Case (based on the "non-union theorem" of L. Blum and M. Blum [BB75]), and first studied by Smith [Smi82]. The main motivation of the work on teams is to investigate how teams of learners can

cooperate to learn collections of computable functions (from their graphs), which collections of functions no individual member of the team can learn alone. We direct the reader to [Smi82, Pit89, JS90b] for description of scenarios involving teams of learners. There is an interesting connection between identification by a team and identification by a probabilistic learner. The notion of probabilistic learner is due to Freivalds [Fre79]. We next give an informal description of probabilistic learners.

A probabilistic learner behaves very much like a deterministic learner except that every now and then it has the ability to base its actions on the outcome of a random event like a coin flip. (For a discussion of probabilistic Turing machines see Gill [Gil77].) Let p be such that $0 \leq p \leq 1$. A probabilistic learner, $\mathbf{P}$, identifies a function f in the limit with probability p just in case $\mathbf{P}$ identifies f in the limit with probability of success at least p, where the probability is taken over all possible coin flips of $\mathbf{P}$. Pitt [Pit89] showed that for any identification criteria the collection of functions that can be identified by teams of n learners requiring at least 1 to be successful can also be learned (according to the same criterion of success) by a probabilistic learner with probability at least $1/n$. Interestingly, for the success criterion identification in the limit, Pitt [Pit89] also established that the converse of the above result holds. He showed that for any positive integer n and any probability p, if $1/(n+1) < p \leq 1/n$, then the classes of computable functions that can be learned in the limit by a single probabilistic machine with probability p are exactly the same as the classes of computable functions that can be learned in the limit by a team of n deterministic machines, at least one of which is required to be successful. As a consequence of this result, Pitt and Smith [PS88] showed that for team identification in the limit of computable functions, introducing redundancy in the team does not yield any additional learning power. In other words, for all $k > 0$, multiplying both the number of learners in the team and the number of learners required to be successful by k, does not increase the learning ability of the team for identification in the limit of computable functions.

The present paper investigates the above questions for a more practical learning criterion than learning in the limit, namely, *finite identification*. In this setting, a machine, fed the graph of a computable function f, outputs a (possibly empty) sequence of programs. The machine is said to *finitely identify* f just in case its first conjecture (which should exist) is a program for f and the learner sticks to this first conjecture[1]. Finite identification of functions by a probabilistic machine was first studied by Freivalds [Fre79]. In this paper, we study finite identification of functions by teams. The present study of this problem has many interesting features. First, as a consequence of our results, an analog of the above mentioned Pitt's connection does not hold for team finite identification and probabilistic finite identification. This is established by showing that there exists a success ratio at which introducing redundancy in the team does yield extra learning ability. Second, the techniques used in the study of finite identification turn out to be a lot more complex than the ones required in the study of limiting identification.

We now give an informal description of the notions and results discussed in the present

[1] An alternative formulation of finite identification requires the learner to conjecture only one program and that program should be correct. It can easily be seen that this later formulation is equivalent to the former, as a learner can keep repeating its first conjecture.

paper.

A learning machine may be thought of as an algorithmic device that takes as input finite initial sequences of graphs of computable functions and that from time to time conjectures computer programs as hypotheses. The two criteria of success for a learning machine to be successful on a function discussed above are introduced next.

A machine $\mathbf{M}$ is said to $\mathbf{Ex}$-*identify* a recursive function f just in case $\mathbf{M}$, fed the graph of f in any order[2], outputs a sequence of programs that converges to a correct program for f. This criterion of success is essentially identification in the limit introduced by Gold [Gol67]. $\mathbf{Ex}$ is defined to be the class of sets $\mathcal{S}$ of computable functions such that some machine $\mathbf{Ex}$-identifies each function in $\mathcal{S}$. Intuitively, $\mathbf{Ex}$ provides a set theoretic summary of the capability of machines to $\mathbf{Ex}$-identify entire classes of computable functions.

A machine $\mathbf{M}$ is said to $\mathbf{Fin}$-*identify* a computable function f just in case $\mathbf{M}$, fed the graph of f in any order[3], scans a finite portion of the graph such that its first conjecture is a program for f and it never abandons this first conjecture. $\mathbf{Fin}$ is defined to be the class of sets $\mathcal{S}$ of computable functions such that some machine $\mathbf{Fin}$-identifies each function in $\mathcal{S}$. This criterion of success is also referred to as *finite identification*. Finite identification of functions was first studied by Gold [Gol67] and Trakhtenbrot and Barzdin [TB70]. Finite identification may also be thought of as a special case of identification in the limit where the learning machine is not allowed any mind changes (Case and Smith [CS83]).

We now define finite identification by teams of machines. A team of learning machines is a multiset of learning machines. Let m and n range over the set of positive integers such that $m \leq n$. A team of n machines $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n$ is said to $\mathbf{Team}_n^m\mathbf{Fin}$-*identify* a set of recursive functions $\mathcal{S}$ iff (by definition) for each $f \in \mathcal{S}$, there exist $i_1, i_2, \ldots, i_m$, where $1 \leq i_1 < i_2 \cdots < i_m \leq n$, such that each of $\mathbf{M}_{i_1}, \mathbf{M}_{i_2}, \ldots, \mathbf{M}_{i_m}$, $\mathbf{Fin}$-identifies f. $\mathbf{Team}_n^m\mathbf{Fin}$ is defined to be the class of sets $\mathcal{S}$ of recursive functions such that some team of n machines $\mathbf{Team}_n^m\mathbf{Fin}$-identifies each function in $\mathcal{S}$.

One can similarly define the criterion $\mathbf{Team}_n^m\mathbf{Ex}$-*identification* and its set theoretic summary $\mathbf{Team}_n^m\mathbf{Ex}$; these definitions can be traced back to Osherson, Stob, and Weinstein [OSW86] and appeared in Pitt and Smith [PS88]. For both $\mathbf{Team}_n^m\mathbf{Ex}$ and $\mathbf{Team}_n^m\mathbf{Fin}$, we refer to the ratio $\frac{m}{n}$ as the *success ratio* of the team criterion.

In the context of identification in the limit, Pitt and Smith [PS88] showed the following result for teams with success ratio $\frac{1}{2}$, thereby implying that for success ratio $\frac{1}{2}$, introducing redundancy in the team does not yield any extra learning ability. For $m \geq 1$,

$$\mathbf{Team}_{2m}^m\mathbf{Ex} = \mathbf{Team}_2^1\mathbf{Ex}$$

With a view to check if an analog of Pitt's connection holds for finite identification, we consider finite identification of functions by teams with success ratio $\frac{1}{2}$. As a contrast to the identification in the limit case, we show the following.

$$\mathbf{Team}_2^1\mathbf{Fin} \subset \mathbf{Team}_4^2\mathbf{Fin}$$

[2]For the subject of this paper it suffices to consider only canonical order $((0, f(0)), (1, f(1)), \ldots)$ of the graph without any loss of generality.

[3]Again, without loss of generality, it suffices to consider only the canonical order.

The above result implies that for team finite identification with success ratio $\frac{1}{2}$, there is a situation in which redundancy does yield extra learning power. However, the following two results, which settle the question of team finite identification with success ratio $\frac{1}{2}$, show that there is no further gain to be achieved by introducing more redundancy. For $m \geq 1$,

$$\mathbf{Team}_{4m}^{2m}\mathbf{Fin} = \mathbf{Team}_{4}^{2}\mathbf{Fin}$$

$$\mathbf{Team}_{4m+2}^{2m+1}\mathbf{Fin} = \mathbf{Team}_{2}^{1}\mathbf{Fin}$$

Results for success ratios $> 1/2$ are also presented (these results are based on [Fre79] and [DPVW91]).

We now proceed formally. In Section 2, we introduce the notation and define the learning paradigms discussed in this paper. In Section 3, we present our results.

2 Preliminaries

2.1 Notations

Recursion-theoretic concepts not explained below are treated in [Rog67]. N denotes the set of natural numbers, $\{0, 1, 2, 3, \ldots\}$. N^+ denotes the set of positive integers, $\{1, 2, 3, \ldots\}$. $\mathcal{R}$ denotes the class of all *recursive* functions, *i.e.*, total computable functions with arguments and values from N. Unless otherwise specified, a, e, i, j, k, l, m, n, p, q, r, s, t, w, x, y, z, with or without decorations, (decorations are the subscripts, superscripts, primes and the like), range over N. Unless otherwise specified, f, g, and h, with or without decorations, range over $\mathcal{R}$. S and P, with or without decoration range over sets. $\mathcal{S}$ and $\mathcal{C}$, with or without decorations, range over subsets of $\mathcal{R}$. η with or without decoration ranges over possibly partial functions.

The symbol $\subseteq$ denotes the subset relation, and $\subset$ denotes proper subset. $\emptyset$ denotes the empty set. The cardinality of a set S is denoted by $\mathrm{card}(S)$. By $\mathrm{card}(S) < \infty$, we mean S is a finite set. $\max(\cdot), \min(\cdot)$, respectively denote the maximum and minimum of a set. By convention $\max(\emptyset) = 0$ and $\min(\emptyset) = \infty$.

φ denotes a fixed acceptable programming system [Rog67, MY78]. φ_i denotes the partial recursive function computed by the i^{th} program in the φ-system. Φ denotes an arbitrary Blum [Blu67] complexity measure associated with acceptable programming system φ.

$\langle \cdot, \cdot \rangle$, denoting a one to one pairing function, is a bijection from $N \times N$ to N. For a real number x, $\lfloor x \rfloor$ denotes the largest integer $\leq x$.

2.2 Finite Function Identification

An information sequence is a mapping from N or an initial segment of N into N. We let SEQ denote the set of all finite information sequences (i.e., information sequences with domain an initial segment of N). We let σ and τ, with or without decorations, range over SEQ. $|\sigma|$ denotes the length of σ.

For a (partial) function η, which is defined for all $x < n$, $\eta[n]$ denotes the finite sequence σ, of length n, such that, for $x < n$, $\sigma(x) = \eta(x)$.

We say that $\sigma \in \text{SEQ}$ is *consistent* with $\tau \in \text{SEQ}$ just in case, for all $x < \min(\{|\sigma|, |\tau|\})$, $\sigma(x) = \tau(x)$.

We say that $\sigma \in \text{SEQ}$ is *consistent* with $f \in \mathcal{R}$ just in case for all $x < |\sigma|$, $\sigma(x) = f(x)$.

For an algorithmic machine $\mathbf{M}$, $\mathbf{M}(\sigma)$ denotes the last conjecture output by $\mathbf{M}$ by the time it has received input σ. If $\mathbf{M}$ has not output any conjecture by the time it has received σ, then we let $\mathbf{M}(\sigma) = \perp$. Hence, a learning machine can be considered as a mapping from SEQ into $N \cup \{\perp\}$. The following definition states this formally. We often use the terms "learning machine" or just "machine" instead of Gold's terminology of inductive inference machine.

Definition 1 [Gol67]

1. A *learning machine* is an algorithmic device that computes a mapping from SEQ into $N \cup \{\perp\}$. We let $\mathbf{M}$, with or without decorations, range over learning machines.
2. For $\sigma \in \text{SEQ}$, $\mathbf{M}(\sigma)$ denotes $\mathbf{M}$'s conjecture on the finite information sequence σ.
3. We further assume that for all $\sigma \subseteq \tau$, if $\mathbf{M}(\sigma) \neq \perp$, then $\mathbf{M}(\tau) \neq \perp$.

Intuitively, $\perp$ is a nonnumeric element that a machine issues to say that it does not wish to conjecture a hypothesis, that is, $\mathbf{M}(\sigma) = \perp$ denotes that $\mathbf{M}$ on σ does not output a conjecture. In the sequel, all reference to a machine means a reference to a learning machine.

We now describe what it means for a learning machine to finitely identify a function.

Definition 2 A learning machine $\mathbf{M}$ is said to **Fin**-*identify* a function f just in case there exists n_0 such that the following hold:

1. for all $n < n_0$, $\mathbf{M}(f[n]) = \perp$;
2. $\varphi_{\mathbf{M}(f[n_0])} = f$; and
3. for all $n \geq n_0$, $\mathbf{M}(f[n]) = \mathbf{M}(f[n_0])$.

If $\mathbf{M}$ **Fin**-identifies f, then we write $f \in \mathbf{Fin}(\mathbf{M})$.

Definition 3 **Fin** denotes the class of all sets $\mathcal{S}$ of recursive functions such that some learning machine **Fin**-identifies each function in $\mathcal{S}$.

2.3 Finite Function Identification by Teams

A team of learning machines is a multiset of learning machines.

Based on the definitions in Smith [Smi82], Pitt [Pit89], and Pitt and Smith [PS88], the following definition of finite identification of functions by teams follows:

Definition 4 Let $m, n \in N^+$, $m \leq n$. A team of machines $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n$ $\mathbf{Team}_n^m\mathbf{Fin}$-*identifies* f (written: $f \in \mathbf{Team}_n^m\mathbf{Fin}(\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n)) \Leftrightarrow \text{card}(\{l \mid 1 \leq l \leq n \ \wedge \ f \in \mathbf{Fin}(\mathbf{M}_l)\}) \geq m$.

$\mathbf{Team}_n^m\mathbf{Fin} = \{\mathcal{S} \mid (\exists \mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n)[\mathcal{S} \subseteq \mathbf{Team}_n^m\mathbf{Fin}(\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n)]\}$.

3 Results

Our results can be divided into two groups. The first group of results is about success ratios greater than $\frac{1}{2}$ and the second group is about success ratio equal to $\frac{1}{2}$. The results in the second group turn out to be considerably more difficult to prove.

Freivalds [Fre79] showed that the learning capability of probabilistic learners above probability $\frac{1}{2}$ is discrete in the sense that if $\frac{n+2}{2n+3} < p \leq \frac{n+1}{2n+1}$, then the collections of functions that can be finitely identified by a probabilistic learner with probability at least p is the same as the collections of functions that can be finitely identified by a probabilistic learner with probability at least $\frac{n+1}{2n+1}$. Freivalds also showed that the collections of functions that can be finitely identified in these discrete intervals form a strict hierarchy. In particular, he showed that the collections of functions that can be finitely identified by a probabilistic learner with probability at least $\frac{n+1}{2n+1}$ is properly contained in the collections of functions that can be finitely identified by a probabilistic learner with probability at least $\frac{n+2}{2n+3}$.

It turns out that using proof techniques similar to Freivalds [Fre79], a similar scenario holds for team identification of functions. Theorem 1, just below, is Freivalds' diagonalization adapted to teams and Theorem 2, due to Daley, Pitt, Velauthapillai, and Will [DPVW91] is Freivalds' simulation adapted to teams.

Theorem 1 $(\forall n \in N)[\mathbf{Team}^{n+2}_{2n+3}\mathbf{Fin} - \mathbf{Team}^{n+1}_{2n+1}\mathbf{Fin} \neq \emptyset]$.

Proof. We give an example collection of functions that witnesses the separation. For $m \leq m'$, consider the class $\mathcal{C}^m_{m'}$ defined as follows.

$\mathcal{C}^m_{m'} = \{f \mid$ the following three conditions are satisfied

(a) $(\forall i \mid 1 \leq i \leq m')[\text{card}(\{x \mid f(\langle i, x\rangle) \neq 0\}) \leq 1]$.

(b) $(\forall i > m')(\forall x)[f(\langle i, x\rangle) = 0]$.

(c) $\text{card}(\{k \mid 1 \leq k \leq m' \ \wedge \ (\exists x)[f(\langle k, x\rangle) \neq 0 \ \wedge \ \varphi_{f(\langle k,x\rangle)} = f]\}) \geq m$.

$\}$

It is easy to see that $\mathcal{C}^{n+2}_{2n+3} \in \mathbf{Team}^{n+2}_{2n+3}\mathbf{Fin}$. Using techniques from Freivalds [Fre79] and Smullyan's [Smu61] multiple recursion theorem, it can be shown that $\mathcal{C}^{n+2}_{2n+3} \notin \mathbf{Team}^{n+1}_{2n+1}\mathbf{Fin}$; we omit the details.

The following corollary to Theorem 1, along with Theorem 2 shows that the highest fraction at which a separation for $\mathbf{Team}^m_n\mathbf{Fin}$-identification from $\mathbf{Fin}$-identification occurs is 2/3.

Corollary 1 $\mathbf{Team}^2_3\mathbf{Fin} - \mathbf{Fin} \neq \emptyset$.

Techniques from [Fre79] were used in [DPVW91] to show that Theorem 1 is tight.

Theorem 2 [DPVW91] *For all r, s such that $\frac{n+2}{2n+3} < \frac{r}{s} \leq \frac{n+1}{2n+1}$,* $\mathbf{Team}^r_s\mathbf{Fin} = \mathbf{Team}^{n+1}_{2n+1}\mathbf{Fin}$.

We now turn our attention to what happens at success ratio 1/2. The following surprising theorem shows that at success ratio 1/2, introducing redundancy in the team results in increased learning power. The theorem states that there are collections of functions that can be finitely identified by teams of 4 learners requiring at least 2 to be successful, but cannot be finitely identified by any team of 2 learners requiring at least one to be successful. As a consequence of this result Pitt's equivalence between limiting identification by probabilistic learners and limiting identification by team learners does not hold for finite identification of functions.

Theorem 3 $\mathbf{Team}_4^2\mathbf{Fin} - \mathbf{Team}_2^1\mathbf{Fin} \neq \emptyset$.

Before we give a formal proof of the above theorem, we present an informal description of the proof. The collections of functions, that will be presented as a witness to the separation, contains functions with the following property: they have program(s) for the function embedded at at least two *special* arguments (out of a maximum of four special arguments). We refer to this class, formally defined later, as $\mathcal{C}_4^2$. It will be easy to see that $\mathcal{C}_4^2 \in \mathbf{Team}_4^2\mathbf{Fin}$ (since the four machines can be asked to output the programs embedded at the special arguments, and by the construction of $\mathcal{C}_4^2$, at least two will be successful.)

Suppose by way of contradiction a team of two machines $\mathbf{M}_1$ and $\mathbf{M}_2$ $\mathbf{Team}_2^1\mathbf{Fin}$-identifies $\mathcal{C}_4^2$. We now give an informal argument demonstrating that there exists a function in $\mathcal{C}_4^2$ on which both $\mathbf{M}_1$ and $\mathbf{M}_2$ fail.

Four programs e_1, e_2, e_3, and e_4 are defined using Smullyan's [Smu61] multiple recursion theorem in such a way that the function computed by at least one of them will be from the class $\mathcal{C}_4^2$ and on which both $\mathbf{M}_1$ and $\mathbf{M}_2$ will fail.

Let f_1, f_2, f_3, and f_4 denote the (partial) functions computed by the programs e_1, e_2, e_3, and e_4, respectively. The idea of the construction is as follows. Initially, programs e_1 and e_2 are embedded at the two special arguments for both f_1 and f_2. Then the following two processes are performed in parallel until the condition in the first step is successful.

1. find if either machine $\mathbf{M}_1$ or $\mathbf{M}_2$ conjectures an output on the f_1 constructed so far;
2. keep extending both f_1 and f_2 such that both are equal on the arguments defined.

If the first condition is never satisfied, then both f_1 and f_2 will be the same and both $\mathbf{M}_1$ and $\mathbf{M}_2$ fail to identify them. Hence, suppose the condition in the first step becomes true. Further suppose that it is machine $\mathbf{M}_1$ (a similar argument can be worked out if it is machine $\mathbf{M}_2$) that conjectures an output for f_1.

Now, f_1 and f_2 are explicitly made different from each other. This ensures that at least one of f_1 defined so far or f_2 defined so far does not agree with $\mathbf{M}_1$'s conjecture.

We first consider the case of $\mathbf{M}_1$'s conjecture not agreeing with f_1 defined so far. (The other case is similar). Program e_3 is embedded at the (third) special argument for f_1, and f_3 is made equal to f_1 defined so far. Then an attempt is made to find if machine $\mathbf{M}_2$ outputs a conjecture on f_1 or f_3 defined so far. This is achieved in a manner similar to the one before by performing the following two processes in parallel until the condition in the first step is successful.

1. find if machine $\mathbf{M}_2$ conjectures an output on f_1 constructed so far;

2. keep extending both f_1 and f_3 such that both are equal on the arguments defined.

Now, if the condition in the first step is never successful then f_1 and f_3 are equal and neither $\mathbf{M}_1$ nor $\mathbf{M}_2$ identifies f_1 ($\mathbf{M}_1$'s conjecture on f_1 is incorrect and $\mathbf{M}_2$ fails to conjecture anything on f_1).

Hence, suppose that the condition in the first step is successful. Then f_1 and f_3 are made distinct from each other and it is ensured that both are members of $\mathcal{C}_4^2$ (by embedding a program at the fourth special argument for f_1 and f_3). Now, clearly $\mathbf{M}_1$ fails to identify both f_1 and f_3. Also, $\mathbf{M}_2$ makes the same conjecture on both f_1 and f_3, and hence fails to identify at least one of f_1 or f_3.

The special arguments for the embedding of the programs come from the cylinders $\{\langle i, x\rangle \mid x \in N\}$, for $1 \leq i \leq 4$. We use the cylinder $\{\langle 0, x\rangle \mid x \in N\}$ for making the functions f_1, f_2, f_3, f_4 different (if needed). The formal description of the proof follows:

PROOF. (Theorem 3) Consider the class $\mathcal{C}_4^2$ defined as follows.

$\mathcal{C}_4^2 = \{f \mid$ the following three conditions are satisfied

(a) $(\forall i \mid 1 \leq i \leq 4)[\text{card}(\{x \mid f(\langle i, x\rangle) \neq 0\}) \leq 1]$.

(b) $(\forall i > 4)(\forall x)[f(\langle i, x\rangle) = 0]$.

(c) $\text{card}(\{k \mid 1 \leq k \leq 4 \ \wedge \ (\exists x)[f(\langle k, x\rangle) \neq 0 \ \wedge \ \varphi_{f(\langle k,x\rangle)} = f]\}) \geq 2$.

$\}$

It is easy to see that $\mathcal{C}_4^2 \in \mathbf{Team}_4^2\mathbf{Fin}$. Suppose by way of contradiction that $\mathcal{C}_4^2 \in \mathbf{Team}_2^1\mathbf{Fin}$ as witnessed by $\mathbf{M}_1$ and $\mathbf{M}_2$. Then by implicit use of the 4-ary recursion theorem [Smu61], there exist distinct e_1, e_2, e_3, e_4 (each e_k, $1 \leq k \leq 4$, greater than 0) such that φ_{e_k} may be described as follows.

Begin φ_{e_k}, $1 \leq k \leq 4$.

1. For $1 \leq i, j \leq 2$, let $\varphi_{e_i}(\langle j, 0\rangle) = e_j$.
   Let $y = \max(\{\langle 1, 0\rangle, \langle 2, 0\rangle\})$.
   For $x \leq y$, $1 \leq i \leq 2$, such that $\varphi_{e_i}(x)$ has not been defined until now let, $\varphi_{e_i}(x) = 0$.
2. **repeat**
   2.1. If there exists an $i \in \{1, 2\}$ such that $\mathbf{M}_i(\varphi_{e_1}[y]) \neq \perp$ then go to step 3.
   2.2. Let $y = y + 1$. For $1 \leq i \leq 2$, let $\varphi_{e_i}(y) = 0$.
   forever
3. Without loss of generality assume that i found in step 2.1 is 1 (otherwise just swap $\mathbf{M}_1$ and $\mathbf{M}_2$).
4. For odd x, such that $\langle 0, x\rangle > y$, let $\varphi_{e_1}(\langle 0, x\rangle) = 1$ and $\varphi_{e_2}(\langle 0, x\rangle) = 2$.
   Note that the above step ensures that at least one of φ_{e_1} (defined until now) or φ_{e_2} (defined until now) is not contained in $\varphi_{\mathbf{M}_1(\varphi_{e_1}[y])}$, though we cannot know effectively

which one. Due to this we need to work separately for each of the possibilities. Below in steps 5 to 7, we describe the process for the possibility when φ_{e_1} is not contained in $\varphi_{\mathbf{M}_1(\varphi_{e_1}[y])}$. A similar process should also be executed in parallel for the possibility when φ_{e_2} is not contained in $\varphi_{\mathbf{M}_1(\varphi_{e_1}[y])}$. This can be done by replacing e_1, e_3 by e_2, e_4 respectively (and with separate local variables) in the steps 5 to 7. We omit the details.

5. Let z be the least x such that $\varphi_{e_1}(\langle 3, x\rangle)$ has not been defined until now. Let $\varphi_{e_1}(\langle 3, z\rangle) = e_3$.

   For x such that $\varphi_{e_1}(x)$ has been defined until now, let $\varphi_{e_3}(x) = \varphi_{e_1}(x)$.

6. **repeat**

   6.1. Suppose z is the least x such that $\varphi_{e_1}(x)$ has not been defined until now.

   6.2. Let $\varphi_{e_1}(z) = \varphi_{e_3}(z) = 0$.

   6.3. If $\mathbf{M}_2(\varphi_{e_1}[z]) \neq \perp$ then go to step 7.

   forever

7. For all x, such that $\varphi_{e_1}(\langle 0, x\rangle)$ has not been defined until now, let $\varphi_{e_1}(\langle 0, x\rangle) = 1$, and $\varphi_{e_3}(\langle 0, x\rangle) = 2$.

   Let x be such that $\varphi_{e_1}(\langle 4, x\rangle)$ has not been defined until now. Let $\varphi_{e_1}(\langle 4, x\rangle) = e_1$. Let $\varphi_{e_3}(\langle 4, x\rangle) = e_3$.

   For all x such that $\varphi_{e_1}(x)$ has not been defined until now let $\varphi_{e_1}(x) = \varphi_{e_3}(x) = 0$.

End φ_{e_k}, $1 \leq k \leq 4$.

Now consider the following cases.

Case 1: The If-clause at step 2.1 is never true.

In this case, let $f = \varphi_{e_1} = \varphi_{e_2} \in \mathcal{C}_4^2$. However, none of $\mathbf{M}_1$ and $\mathbf{M}_2$ outputs a program on f.

Case 2: The If-clause at step 2.1 eventually becomes true.

Let y be as in step 2.1, when the if clause succeeds. Without loss of generality suppose it was $\mathbf{M}_1$ which outputs a program on $\varphi_{e_1}[y]$. Suppose $\varphi_{\mathbf{M}_1(\varphi_{e_1}[y])}$ does not contain φ_{e_1} defined until the end of step 4 (case of $\varphi_{\mathbf{M}_1(\varphi_{e_1}[y])}$ does not contain φ_{e_2} is similar; see discussion at the end of step 4). We now consider the following cases.

Case 2.1 The If-clause at step 6.3 is never true.

In this case let $f = \varphi_{e_1} = \varphi_{e_3} \in \mathcal{C}_4^2$. Note that $\mathbf{M}_2$ does not output a program on f, and by discussion above the program output by $\mathbf{M}_1$ on f does not compute f.

Case 2.2 The If-clause at step 6.3 eventually becomes true.

In this case let $f = \varphi_{e_1} \in \mathcal{C}_4^2$ and $f' = \varphi_{e_3} \in \mathcal{C}_4^2$. Note that both f and f' belong to $\mathcal{C}_4^2$ and are distinct from each other. Note that the program output by $\mathbf{M}_1$ on f and f' is the same and does not compute either of them. Also the program output by $\mathbf{M}_2$ on f and f' is the same and, since f and f' are distinct, it does not compute at least one of them.

From the above cases we have that $\mathcal{C}_4^2$ is not $\mathbf{Team}_2^1\mathbf{Fin}$-identified by $\mathbf{M}_1, \mathbf{M}_2$. ∎

As a contrast to the above Theorem 3, we now present the two surprising Theorems 4 and 5 below. The proof of these two theorems involves a complex simulation argument. We first introduce some technical machinery that will be useful in proving these theorems.

Definition 5 Suppose $\mathbf{M}$ is a machine and η is a (partial) function. Then $\mathrm{ConjPoint}(\mathbf{M}, \eta) = \min(\{n \mid (\forall x < n)[\eta(x)\downarrow] \ \wedge \ \mathbf{M}(\eta[n]) \neq \perp\})$.

Intuitively, ConjPoint denotes the point at which $\mathbf{M}$, fed η, outputs its first conjecture; $\mathrm{ConjPoint}(\mathbf{M}, \eta)$ is ∞ if $\mathbf{M}$ does not output a conjecture on any initial segment of η.

We motivate the next definition. Consider a set of programs P. Let $i \leq \mathrm{card}(P)$ be given. Also, suppose we are given the initial segment $f[m]$ of some recursive function f. It is useful to find if there are i distinct programs in the set P that compute an extension of $f[m]$. In particular, we will be given an upper bound s, and we would like to find the maximum number $r \leq s$ such that there exists a subset S of P of cardinality i with the following properties:

- each program in S extends $f[m]$,
- each program in S is defined on arguments $< r$. Moreover, all programs in S compute the same value on arguments $< r$,
- the above two conditions can be checked in $\leq s$ steps.

The function Maxcons defined below formalizes this notion.

Definition 6 Suppose P is a finite set of programs, $i \leq \mathrm{card}(P)$ and f is a total function. Then,

$\mathrm{Maxcons}(P, i, s, f[m]) = \max(\{r \leq s \mid \exists S \subseteq P$ such that the following four conditions are satisfied

$\mathrm{card}(S) = i$,
$(\forall p \in S)(\forall x < \max(m, r))[\Phi_p(x) \leq s]$,
$(\forall p \in S)(\forall x < m)[f(x) = \varphi_p(x)]$, and
$(\forall p, p' \in S)(\forall x < r)[\varphi_p(x) = \varphi_{p'}(x)]$

$\}$).

Intuitively, Maxcons determines the maximal initial consistency among subsets S (of size i) of P, as can be judged using a time bound of s, such that S contains only programs that extend $f[m]$. Once, Maxcons has been used to determine the point of maximal consistency, we would like to find one such set S. The function Progcons defined below returns the lexicographically least subset S of P. (By lexicographical order, we mean the dictionary order.)

Definition 7 Suppose P is a finite set of programs, $i \leq \mathrm{card}(P)$ and $f \in \mathcal{R}$. Then,

$\mathrm{Progcons}(P, i, s, f[m], n) =$ lexicographically least subset, S, of P of cardinality i, if any, such that the following conditions are satisfied

$(\forall p \in S)(\forall x < \max(m, n))[\Phi_p(x) \leq s]$,
$(\forall p \in S)(\forall x < m)[f(x) = \varphi_p(x)]$, and
$(\forall p, p' \in S)(\forall x < n)[\varphi_p(x) = \varphi_{p'}(x)]$.

We always use Progcons in conjunction with Maxcons with the parameter n in Progcons having the value of Maxcons$(P, i, s, f[m])$. It should be noted that if Maxcons$(P, i, s, f[m]) = 0$, then Progcons$(P, i, s, f[m], 0)$ may not be defined. In such cases, we take, by convention, the value of Progcons to be an arbitrary subset of P of size i. This is merely for ease of presenting the proof.

We now describe a procedure, which aids in separating functions computed by sufficiently large fraction of programs from a set of programs.

Suppose $m \leq m'$, an initial segment $f[m']$, and a set P, card$(P) < 3j$, of programs is given. Further suppose that at least j programs in P compute (partial) functions that extend $f[m']$. Note that there can be at most two distinct total functions which are computed by at least j of the programs in P. The aim of the following procedure is to effectively find the functions, if any, which extend $f[m]$ and are computed by at least j programs in P; f_1, f_2 in the definition of Simul denote these two functions (if any).

Simul$(P, j, f[m'], m, f_1, f_2)$

Assumptions about the input: (a) $m \leq m'$, (b) card$(P) < 3j$, and (c) $f[m']$ is an initial segment of at least j programs in P.

f_1, f_2 denote the functions (or, initial segments of functions) extending $f[m]$, if any, that are computed by at least j programs in P.

1. For $x < m$, let $f_1(x) = f(x)$.

   For $x < m'$, let $f_2(x) = f(x)$.

   Let t be such that there exists a set $S \subseteq P$ of cardinality j, such that $(\forall i \in S)(\forall x < m')[\Phi_i(x) \leq t \ \wedge \ \varphi_i(x) = f(x)]$. (Note that by assumption such a t exists since the functions computed by at least j programs in the set P contain $f[m']$ as an initial segment). Go to stage t. (Note that there are no stages less than t; this is merely for convenience of writing the construction).

   Stage s

   1. Let $m_1 = \text{Maxcons}(P, j, s, f[m])$.
   2. Let $S = \text{Progcons}(P, j, s, f[m], m_1)$.
   3. Let w be an element of S. If f_2 defined until now is consistent with $\varphi_w[m_1]$, then let $i = 2$ and $i' = 1$; else let $i = 1$ and $i' = 2$.
   4. For $x < m_1$, let $f_i(x) = \varphi_w(x)$.
   5. Let $m_2 = \text{Maxcons}(P - S, j, s, f[m])$.
   6. If $m_2 = 0$, then let S' be an arbitrary subset of P, of size j; else let $S' = \text{Progcons}(P - S, j, s, f[m], m_2)$.

      (Note that if less than j programs in $P - S$ are found to be extending $f[m]$, then $m_2 = 0$.)
   7. Let w' be an element of S'.
   8. For $x < m_2$, let $f_{i'}(x) = \varphi_{w'}(x)$.
   9. Go to Stage $s + 1$.

   End stage s.

It is easy to show using induction on the stages that the following lemma regarding Simul holds.

Lemma 1 *Suppose $j \in N^+$, a set P of cardinality less than $3j$, $m \leq m'$, and an initial segment $f[m']$ are given. Further suppose that at least j of the programs in P compute functions that extend $f[m']$. Then, for f_1, f_2, as defined in* $\text{Simul}(P, j, f[m'], m, f_1, f_2)$*, the following hold:*

(a) $f[m'] \subseteq f_2$,

(b) $f[m] \subseteq f_1$, *and*

(c) $(\forall g \supseteq f[m] \mid g \in \mathcal{R} \ \wedge \ \text{card}(\{i \in P \mid \varphi_i = g\}) \geq j)[f_1 = g \ \vee \ f_2 = g]$.

We now present our first simulation result which implies that for all positive *odd* numbers m, the collections of functions that can be finitely identified by teams of $2m$ machines requiring at least m to be successful can also be identified by a team of two machines requiring at least one to be successful.

Theorem 4 $\mathbf{Team}^{2j+1}_{4j+2}\mathbf{Fin} \subseteq \mathbf{Team}^{1}_{2}\mathbf{Fin}$.

We first give an informal high level description of the simulation required to establish this result. Suppose $\mathcal{C}$ is a collection of functions that can be $\mathbf{Team}^{2j+1}_{4j+2}\mathbf{Fin}$-identified. Let a team consisting of machines $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_{4j+2}$ witness the identification. We then describe machines $\mathbf{M}'_1$ and $\mathbf{M}'_2$ that $\mathbf{Team}^{1}_{2}\mathbf{Fin}$-identify $\mathcal{C}$. Let $f \in \mathcal{C}$.

The machine $\mathbf{M}'_1$ simulates each of the $4j+2$ machines, $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_{4j+2}$, on f and waits for at least $2j+1$ machines to output a conjecture. It then outputs a procedure X that depends on these $2j+1$ conjectures and the convergence point of the $2j+1$ machines on f.

The machine $\mathbf{M}'_2$ also simulates each of the $4j+2$ machines, $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_{4j+2}$, on f but waits for at least $3j+2$ machines to output a conjecture. It then outputs a procedure Y that depends on these $3j+2$ conjectures and the convergence point of the $3j+2$ machines on f.

The procedures X and Y cooperate to make sure that at least one of them computes f. This is ensured as follows: Procedure X keeps track of whether procedure Y gets initiated or not. The cooperation between the two procedures begins when it is discovered that there are at least $j+1$ programs in the first $2j+1$ conjectures that agree with a suitable initial segment of f. Following case analysis gives a high level description of which procedure simulates the function in which case.

Case 1: Less than $3j+2$ machines in the team consisting of machines $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_{4j+2}$ output a conjecture on f. In this case, there are at least $j+1$ correct programs for f in the conjectures of first $2j+1$ machines. Procedure X uses procedures Maxcons and Progcons to find these programs and simulates f.

Case 2: At least $3j+2$ machines in team consisting of machines $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_{4j+2}$ output a conjecture on f. There are two subcases:

Case 2.1: There are less than $j+1$ programs in the first $2j+1$ conjectures that compute a suitable initial segment of f. In this case cooperation between procedure Y and X will

not take place and procedure Y will use the procedures Maxcons and Progcons to find $j+1$ correct programs from the first $3j+2$ programs and simulate f.

Case 2.2: There are at least $j+1$ programs in the first $2j+1$ conjectures that compute a suitable initial segment of f. In this case both procedures X and Y collaborate and use the procedure Simul to ensure that at least one of them simulates f.

We now proceed formally.

PROOF. Suppose machines $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_{4j+2}$ $\mathbf{Team}^{2j+1}_{4j+2}\mathbf{Fin}$-identify $\mathcal{C}$. We construct machines $\mathbf{M}'_1, \mathbf{M}'_2$ which $\mathbf{Team}^{1}_{2}\mathbf{Fin}$-identify $\mathcal{C}$. We assume without loss of generality that, for all functions f, $\text{ConjPoint}(\mathbf{M}_1, f) \leq \text{ConjPoint}(\mathbf{M}_2, f) \leq \cdots \leq \text{ConjPoint}(\mathbf{M}_{4j+2}, f)$. We further assume that, for any function f, the programs output by different machines, if any, are different (otherwise we can easily ensure this by padding).

Fix f. Let the programs output (if any), on f, by machines $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_{3j+2}$, be $p_1, p_2, \ldots, p_{3j+2}$, respectively. $\mathbf{M}'_1$, fed f, waits until machine $\mathbf{M}_{2j+1}$ has output its guess. $\mathbf{M}'_1$ then outputs the program $X(p_1, p_2, \ldots, p_{2j+1}, f[\text{ConjPoint}(\mathbf{M}_{2j+1}, f)])$ (where $X(\cdots)$ is defined below). $\mathbf{M}'_2$, fed f, waits until machine $\mathbf{M}_{3j+2}$ has output its guess. $\mathbf{M}'_2$ then outputs the program $Y(p_1, p_2, \ldots, p_{2j+1}, p_{2j+2}, p_{2j+3}, \ldots, p_{3j+2}, f[\text{ConjPoint}(\mathbf{M}_{3j+2}, f)])$ (where $Y(\cdots)$ is defined below).

We describe algorithms for X and Y below. For convenience of presentation, we describe the algorithm for Y first.

begin $\varphi_{Y(p_1,p_2,\ldots,p_{2j+1},p_{2j+2},p_{2j+3},\ldots,p_{3j+2},f[m])}$

(For notational convenience, we will use φ_Y below instead of $\varphi_{Y(p_1,p_2,\ldots,p_{3j+2},f[m])}$. Note that for the case of interest $m = \text{ConjPoint}(\mathbf{M}_{3j+2}, f)$).

1. For $x < m$, let $\varphi_Y(x) = f(x)$.
2. Let k be the least value, if any, such that, there exists a set $S \subseteq \{p_1, p_2, \ldots, p_{2j+1}, p_{2j+2}, p_{2j+3}, \ldots, p_{3j+2}\}$, of cardinality $j+1$, such that $(\forall p \in S)(\forall x < m)[\Phi_p(x) \leq k \ \wedge \ \varphi_p(x) = f(x)]$. Go to stage $k+1$ (note that there are no stages less than $k+1$. This is just for the ease of presentation).

   Begin stage s

   2.1. If there exists a subset S of $\{p_1, p_2, \ldots, p_{2j+1}\}$ of cardinality $j+1$, such that $(\forall p \in S)(\forall x < m)[\Phi_p(x) \leq s \ \wedge \ \varphi_p(x) = f(x)]$, then go to step 3.

   2.2. Let $m_0 = \text{Maxcons}(\{p_1, \ldots, p_{3j+2}\}, j+1, s, f[m])$.

   2.3. Let $S = \text{Progcons}(\{p_1, \ldots, p_{3j+2}\}, j+1, s, f[m], m_0)$.

   2.4. Let w be an element of S. For $x < m_0$, let $\varphi_Y(x) = \varphi_w(x)$.

   2.5. Go to stage $s+1$.

   End stage s
3. Proceed to collaborate with $\varphi_{X(p_1,p_2,\ldots,p_{2j+1},f[n])}$ (at step 3), where $n = \text{ConjPoint}(\mathbf{M}_{2j+1}, f)$, as described in the procedure for $\varphi_X(p_1, p_2, \ldots, p_{2j+1}, f[n])$ below.

end $\varphi_{Y(p_1,p_2,\ldots,p_{2j+1},p_{2j+2},p_{2j+3}\ldots,p_{3j+2},f[m])}$.

begin $\varphi_{X(p_1,p_2,\ldots,p_{2j+1},f[n])}$

(For notational convenience, we will use φ_X below instead of $\varphi_{X(p_1,p_2,\ldots,p_{2j+1},f[n])}$. Note that for the case of interest $n = \text{ConjPoint}(\mathbf{M}_{2j+1}, f)$).

1. Let t be such that there exists a subset S of $\{p_1, p_2, \ldots, p_{2j+1}\}$ of cardinality $j+1$, such that $(\forall x < n)(\forall p \in S)[\Phi_p(x) \leq t \ \wedge \ \varphi_p(x) = f(x)]$. Go to stage t. Note that there are no stages $< t$. This is just for the ease of writing the proof. If no such t exists then φ_X is the everywhere undefined function.

   Begin stage s

   1.1. Let $m_1 = \text{Maxcons}(\{p_1, p_2, \ldots, p_{2j+1}\}, j+1, s, f[0])$.

   1.2. Let $S = \text{Progcons}(\{p_1, p_2, \ldots, p_{2j+1}\}, j+1, s, f[0], m_1)$.

   1.3. Let w be an element of S. If $\text{ConjPoint}(\mathbf{M}_{3j+2}, \varphi_w) \leq m_1$, then go to step 2.

   1.4. For all $x < m_1$, let $\varphi_X(x) = \varphi_w(x)$.

   1.5. Go to stage $s+1$.

   End stage s.

2. Let $m_2 = \text{ConjPoint}(\mathbf{M}_{3j+2}, \varphi_w)$.

   For all $x < m_2$, define $\varphi_X(x) = \varphi_w(x)$.

3. Suppose the output of machines $\mathbf{M}_{2j+2}, \mathbf{M}_{2j+3}, \ldots, \mathbf{M}_{3j+2}$ on $\varphi_w[m_2]$ are $p'_{2j+2}, \ldots, p'_{3j+2}$ respectively. For notational convenience we will use φ_Y instead of $\varphi_{Y(p_1,\ldots,p_{2j+1},p'_{2j+2},\ldots,p'_{3j+2},\varphi_w[m_2])}$ below. Note that if the input function being learned is consistent with $\varphi_w[m_2]$, then φ_Y simulated here is the same as φ_Y which will be defined for the input function. Also, if the current procedure has reached this point, then φ_Y would have recognized that, and would cooperate with φ_X (see the algorithm for φ_Y). Also, φ_X, φ_Y are consistent with each other until this point. Also, φ_X has been defined only for inputs $< m_2$. Also, φ_Y defined until now is contained in the (partial) functions computed by at least $j+1$ programs in $\{p_1, \ldots, p_{2j+1}, p'_{2j+2}, \ldots, p'_{3j+2}\}$. Let m_3 be such that, φ_Y is defined on exactly the inputs $< m_3$, when it starts collaborating with φ_X.

   Let $\varphi_X = f_1$ and $\varphi_Y = f_2$, where f_1 and f_2 are as obtained in $\text{Simul}(\{p_1, p_2, \ldots, p_{3j+2}\}, j+1, \varphi_Y[m_3], m_2, f_1, f_2)$.

end $\varphi_{X(p_1,p_2,\ldots,p_{2j+1},f[n])}$

Now fix f, which is $\mathbf{Team}^{2j+1}_{4j+2}\mathbf{Fin}$-identified by $\mathbf{M}_1, \ldots, \mathbf{M}_{3j+2}$. Suppose $p_1, p_2, p_3, \ldots, p_{3j+2}$ are the programs, if any, output by $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_{3j+2}$ on f, respectively. Recall that according to our assumption regarding the machines, if $\mathbf{M}_{i+1}$ outputs a program on f then it follows that $\mathbf{M}_i$ also outputs a program on f. For ease of notation let X denote $X(p_1, \ldots, p_{2j+1}, f[\text{ConjPoint}(\mathbf{M}_{2j+1}, f)])$, and if $\text{ConjPoint}(\mathbf{M}_{3j+2}, f) \neq \infty$, then let Y denote $Y(p_1, \ldots, p_{3j+2}, f[\text{ConjPoint}(\mathbf{M}_{3j+2}, f)])$.

We prove below that, for $f \in \mathcal{C}$, at least one of X, Y computes f.

There are following cases:

Case 1: $\mathbf{M}_{3j+2}$ does not output a program on f.

In this case, at least $j+1$ of the programs in $\{p_1, \ldots, p_{2j+1}\}$ are programs for f. Since on f, less than $3j+2$ machines output a program, in the procedure for X step 1.3 would

never succeed. It follows that in the simulation at steps 1.1 to 1.5, X will simulate f.

Case 2: At least $3j+2$ machines (of $\mathbf{M}_1, \ldots, \mathbf{M}_{4j+2}$) output a program on f.

Suppose $m = \text{ConjPoint}(\mathbf{M}_{3j+2}, f)$.

Case 2.1: There are less than $j+1$ programs in $p_1, \ldots, p_{2j+1}$ which calculate $f[m]$ correctly.

In this case, step 2.1 of Y will not succeed. Also there exist $j+1$ programs in $\{p_1, \ldots, p_{3j+2}\}$ which compute f correctly. Thus in steps 2.1 to 2.5 (of Y) Y will simulate f.

Case 2.2: There are at least $j+1$ programs in $p_1, \ldots, p_{2j+1}$, which calculate $f[m]$ correctly.

In this case, both X and Y will be cooperating with each other to ensure correctness of at least one of X and Y (due to properties of Simul as noted in Lemma 1). ∎

The next theorem completes the picture for finite identification of functions by teams with success ratio 1/2. It shows that for all positive *even* numbers m, the collections of functions that can be finitely identified by a team of $2m$ machines requiring at least m to be successful can also be finitely identified by a team of 4 machines requiring at least 2 to be successful. The proof of this theorem is a more complicated version of the proof of Theorem 4. To get an informal overview of the simulation, we recommend that the reader peruse through the case analysis at the end of the proof before reading the descriptions of various procedures.

Theorem 5 $(\forall j \in N^+)[\mathbf{Team}_{4j}^{2j}\mathbf{Fin} \subseteq \mathbf{Team}_4^2\mathbf{Fin}]$.

Proof. Let machines $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_{4j}$ $\mathbf{Team}_{4j}^{2j}\mathbf{Fin}$-identify $\mathcal{C}$. Without loss of generality we assume that, for all f, $\text{ConjPoint}(\mathbf{M}_1, f) \leq \text{ConjPoint}(\mathbf{M}_2, f) \leq \ldots \leq \text{ConjPoint}(\mathbf{M}_{4j}, f)$. We further assume, without loss of generality, that for all f, the programs output by different machines, if any, are different. We construct machines $\mathbf{M}'_1, \mathbf{M}'_2, \mathbf{M}'_3$, and $\mathbf{M}'_4$ which $\mathbf{Team}_4^2\mathbf{Fin}$-identify $\mathcal{C}$.

Let the programs output (if any) by machines $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_{4j}$ be $p_1, p_2, \ldots, p_{4j}$ respectively. $\mathbf{M}'_1, \mathbf{M}'_2$, fed a function f, wait until machine $\mathbf{M}_{2j}$ has output its guess. $\mathbf{M}'_1$ then outputs the program $X_1(p_1, p_2, \ldots, p_{2j}, f[n])$, and $\mathbf{M}'_2$ outputs the program $X_2(p_1, p_2, \ldots, p_{2j}, f[n])$, where $n = \text{ConjPoint}(\mathbf{M}_{2j}, f)$. Similarly, $\mathbf{M}'_3$ waits until machine $\mathbf{M}_{3j}$ has output its guess. $\mathbf{M}'_3$ then outputs the program $Y(p_1, p_2, \ldots, p_{3j}, f[m])$, where $m = \text{ConjPoint}(\mathbf{M}_{3j}, f)$. $\mathbf{M}'_4$ waits until machine $\mathbf{M}_{3j+1}$ has output its guess. $\mathbf{M}'_4$ then outputs the program $Z(p_1, p_2, \ldots, p_{3j+1}, f[l])$, where $l = \text{ConjPoint}(\mathbf{M}_{3j+1}, f)$.

We describe X_1, X_2, Y and Z below. X_1, X_2 are described together since they are similar. We do not describe Z separately, but it behaves as described in procedures for Y and X_i. We describe Y first for ease of presentation.

begin $\varphi_{Y(p_1,p_2,\ldots,p_{3j},f[m])}$.

For notational convenience we will write Y instead of $Y(p_1, p_2, \ldots, p_{3j}, f[m])$ below. Note that for the case of interest $m = \text{ConjPoint}(\mathbf{M}_{3j}, f)$. We assume without loss of generality that, $f[m] \subseteq \varphi_{\mathbf{M}_{3j}(f[m])}$.

1. Go to stage 0.

Stage s

1.1 If there exists a subset S of $\{p_1, \ldots, p_{2j}\}$ of cardinality j, such that $(\forall p \in S)(\forall x < m)[\Phi_p(x) < s \ \wedge \ \varphi_p(x) = f(x)]$, then go to step 2.

1.2. If there exists a subset S of $\{p_1, \ldots, p_{3j}\}$ of cardinality j, and r such that

$m \leq r \leq s$,
$(\forall p \in S)(\forall x < r)[\Phi_p(x) < s]$,
$(\forall p \in S)(\forall x < m)[\varphi_p(x) = f(x)]$,
$(\forall p, p' \in S)(\forall x < r)[\varphi_p(x) = \varphi_{p'}(x)]$,
for $p \in S, \text{ConjPoint}(\mathbf{M}_{3j+1}, \varphi_p) \leq r$,

then go to step 3.

1.3. Go to stage $s+1$.

End stage s.

2. For $x < m$, let $\varphi_Y(x) = f(x)$. Proceed to collaborate with $\varphi_{X_i(p_1,\ldots,p_{2j},f[n])}$ (step 3) where $n = \text{ConjPoint}(\mathbf{M}_{2j}, f)$, as described in the procedure for X_i.

3. Let S be as found in step 1.2. Suppose $p \in S$. Let $l' = \text{ConjPoint}(\mathbf{M}_{3j+1}, \varphi_p)$ and $p'_{3j+1} = \mathbf{M}_{3j+1}(\varphi_p[l'])$.

Let Z denote $Z(p_1, \ldots, p_{3j}, p'_{3j+1}, \varphi_p[l'])$.

Let $\varphi_Y = \varphi_Z = f_2$, where f_2 is as in $\text{Simul}(\{p_1, \ldots, p_{3j}, p'_{3j+1}\}, j+1, \varphi_p[l'], m, f_1, f_2)$.

end $\varphi_{Y(p_1,p_2,\ldots,p_{3j},f[m])}$.

begin$\varphi_{X_i(p_1,p_2,\ldots,p_{2j},f[n])}, i = 1, 2$

(For notational convenience we will use φ_{X_i} below instead of $\varphi_{X_i(p_1,p_2,\ldots,p_{2j},f[n])}$. Note that for the case of interest $n = \text{ConjPoint}(\mathbf{M}_{2j}, f)$.)

1. Go to stage 0.

Stage s

1.1. If there exists m_1 and a set $S \subseteq \{p_1, p_2, \ldots, p_{2j}\}$ such that

$n \leq m_1 \leq s$
$\text{card}(S) = j$,
$(\forall p \in S)(\forall x < m_1)[\Phi_p(x) \leq s]$,
$(\forall p \in S)(\forall x < n)[\varphi_p(x) = f(x)]$,
$(\forall i_1, i_2 \in S)(\forall x < m_1)[\varphi_{i_1}(x) = \varphi_{i_2}(x)]$ and
$\text{ConjPoint}(\mathbf{M}_{3j}, \varphi_w) \leq m_1$,

then let S be one such set. Go to step 2.

1.2. Let $m_1 = \text{Maxcons}(\{p_1, p_2, \ldots, p_{2j}\}, j+1, s, f[n])$.

1.3. Let $S = \text{Progcons}(\{p_1, p_2, \ldots, p_{2j}\}, j+1, s, f[n], m_1)$.

1.4. For $x < m_1$, let $\varphi_{X_1}(x) = \varphi_{X_2}(x) = \varphi_w(x), w \in S$.

1.5. Go to Stage $s+1$.

End stage s.

2.1. Let S be as found in step 1.1. Let w be an element of S (Note that this w will be referred to in several places below). Let $m_2 = \text{ConjPoint}(\mathbf{M}_{3j}, \varphi_w)$. For $x < m_2$, let $\varphi_{X_1}(x) = \varphi_w(x)$.

2.2. Let $S' = \{p_1, p_2, \ldots, p_{2j}\} - S$. If all the programs in S' are convergently different from $\varphi_w[m_2]$ and there exists an m_2' such that $(\forall i_1, i_2 \in S')(\forall x < m_2')[\varphi_{i_1}(x) \downarrow = \varphi_{i_2}(x) \downarrow]$ and $\mathbf{M}_{3j}$ outputs a program on $\varphi_{w'}[m_2'], w' \in S'$, then a procedure similar to the one below (from step 2.3 onwards) is started with φ_{X_2} playing the role of φ_{X_1}, m_2' playing the role of m_2, w' playing the role w, and S' playing the role of S. Note that if there does not exist such a m_2' then X_2 is always available for use below (for example if the simulation extends φ_{X_2} in steps 5, 6).

2.3. For $2j < i \leq 3j$, let $p_i' = \mathbf{M}_i(\varphi_w[m_2])$.

Let Y denote $Y(p_1, \ldots, p_{2j}, p_{2j+1}', \ldots, p_{3j}', \varphi_w[m_2])$. Note that if the input function is consistent with $\varphi_w[m_2]$, then Y is same as Y output on the input function.

2.4. If in the procedure for φ_Y, step 1.1. succeeds, then go to step 3.

Else, let S, l' be as in step 3 in the procedure for Y. Let $S_1 = S$. Let $p_{3m+1}' = \mathbf{M}_{3j+1}(\varphi_Y[l'])$. Let $W' = X_1$, $W'' = X_2$. Let w_1 be a member of S_1. Go to step 5.

3. Note that φ_Y and φ_{X_1} defined until now are $\varphi_w[m_2]$. Moreover, at least $j+1$ programs in $\{p_1, \ldots, p_{2j}, p_{2j+1}', \ldots, p_{3j}'\}$ compute extensions of $\varphi_w[m_2]$.

Let t be the last stage executed in step 1 above. Go to stage $t+1$.

Stage s.

3.1. If there exists $m_3 \leq s$ and a set $S_1 \subseteq \{p_1, p_2, \ldots, p_{2j}, p_{2j+1}', \ldots, p_{3j}'\}$ such that
$\mathrm{card}(S_1) = j$,
$m_2 \leq m_3 \leq s$,
$(\forall p \in S_1)(\forall x < m_3)[\Phi_p(x) \leq s]$,
$(\forall x < m_2)(\forall p \in S_1)[\varphi_p(x) = \varphi_w(x)]$.
$(\forall i_1, i_2 \in S_1)(\forall x < m_3)[\varphi_{i_1}(x) = \varphi_{i_2}(x)]$, and
$\mathrm{ConjPoint}(\mathbf{M}_{3j+1}, \varphi_{w_1}) \leq m_3$, for $w_1 \in S_1$,
then let S_1 be one such set. Go to step 4.

3.2. Let $m_3 = \mathrm{Maxcons}(\{p_1, p_2, \ldots, p_{2j}, p_{2j+1}', \ldots, p_{3j}'\}, 2j, s, \varphi_w[m_2])$.

3.3. Let $S_1 = \mathrm{Progcons}(\{p_1, p_2, \ldots, p_{2j}, p_{2j+1}', \ldots, p_{3j}'\}, 2j, s, \varphi_w[m_2], m_3)$.

3.4. For $x < m_3$, let $\varphi_{X_1}(x) = \varphi_Y(x) = \varphi_{w_1}(x)$, where $w_1 \in S_1$.

3.5. Go to Stage $s+1$.

End Stage s.

4. Let S_1 be as in step 3.1. Let $w_1 \in S_1$. Let $l' = \mathrm{ConjPoint}(\mathbf{M}_{3j+1}, \varphi_{w_1})$. Let $p_{3j+1}' = \mathbf{M}_{3j+1}(\varphi_{w_1}[l'])$. Let Z denote $Z(p_1, \ldots, p_{2j}, p_{2j+1}', \ldots, p_{3j+1}', \varphi_{w_1}[l'])$. If φ_Y defined until now is consistent with $\varphi_{w_1}[l']$, then let $W = Y$, $W' = X_1$ and $W'' = X_2$; otherwise let $W = X_2$, $W' = Y$ and $W'' = X_1$. Let $\varphi_W = \varphi_Z = f_2$, where f_2 is as in $\mathrm{Simul}(\{p_1, \ldots, p_{2j}, p_{2j+1}', \ldots, p_{3j+1}'\}, j+1, \varphi_{w_1}[l'], m_2, f_1, f_2)$. Go to step 5.

5. Let $S_1' = \{p_1, p_2, \ldots, p_{2j}, p_{2j+1}', \ldots, p_{3j}'\} - S_1$. Go to stage 0.

Stage s

5.1. If there exists a set $S_2 \subseteq S_1'$ of cardinality $j+1$ such that, $(\forall p \in S_2)(\forall x < l')[\Phi_p(x) \leq s \ \wedge \ \varphi_p(x) = \varphi_{w_1}(x)]$, then go to step 6.1.

5.2. { *This step and step 6.2 guard against the possibility that there may be up to 3 distinct initial segments extending $f[m_2]$ on which $\mathbf{M}_{3j+1}$ outputs a program, and each of these initial segments is extended by at least j programs*

in $\{ p_1, p_2, \ldots, p_{2j}, p'_{2j+1}, \ldots, p'_{3j} \}$*; the reader is advised to also look at Case* 3.2.2 *in the case analysis at the end of this proof* }

If there exists a set $S_2 \subseteq S'_1$ of cardinality j and an $l'' \geq m_2$ such that

$(\forall p \in S_2)[(\forall x < l'')[\Phi_p(x) \leq s]$,

$(\forall x < m_2)[\varphi_p(x) = \varphi_w(x)]]$,

$(\forall p, p' \in S_2)(\forall x < l'')[\varphi_p(x)\downarrow = \varphi_{p'}(x)\downarrow]$ and

$(\exists x < \min(l', l''))[\varphi_p(x) \neq \varphi_{w_1}(x)]$, where $p \in S_2$, and

$\text{ConjPoint}(\mathbf{M}_{3j+1}, \varphi_p) = l''$, where $p \in S_2$,

Then go to step 6.2.

5.3. Let $m_5 = \text{Maxcons}(S'_1, 2j, s, \varphi_w[m_2])$.

5.4. Let $S_2 = \text{Progcons}(S'_1, 2j, s, \varphi_w[m_2], m_5)$.

5.5. For $x < m_5$, let $\varphi_{W'}(x) = \varphi_{W''}(x) = \varphi_{w_2}(x)$, for $w_2 \in S_2$.

5.6. Go to stage $s + 1$.

End stage s.

6.1. Let $\varphi_{W'} = \varphi_{W''} = f_1$, where f_1 is as in $\text{Simul}(\{p_1, \ldots, p_{2j}, p'_{2j+1}, \ldots, p'_{3j+1}\}, j + 1, \varphi_{w_1}[l'], m_2, f_1, f_2)$. HALT.

6.2. Let S_2, l'', be as found in step 5.2. Let w_2 be a member of S_2. Let $p''_{3j+1} = \mathbf{M}_{3j+1}(\varphi_{w_2}[l''])$. Let Z' denote $Z(p_1, \ldots, p_{2j}, p'_{2j+1}, \ldots, p'_{3j}, p''_{3j+1}, \varphi_{w_2}[l''])$. Let $\varphi_{W'} = \varphi_{Z'} = f_2$, where f_2 is as in $\text{Simul}(\{p_1, \ldots, p_{2j}, p'_{2j+1}, \ldots, p'_{3j}, p''_{3j+1}\}, j + 1, \varphi_{w_2}[l''], m_2, f_1, f_2)$.

Let $S_3 = S'_1 - S_2$. If and when it is discovered, that there exists a l''', $(\forall p \in S_3)(\forall x < m_2)[\varphi_p(x) = \varphi_w(x)]$ and $(\forall p, p' \in S_3)(\forall x < l''')[\varphi_p(x)\downarrow = \varphi_{p'}(x)\downarrow]$ and $(\exists x < \min(l', l'''))[\varphi_p(x) \neq \varphi_{w_1}(x)]$, and $(\exists x < \min(l'', l'''))[\varphi_p(x) \neq \varphi_{w_2}(x)]$, and $\text{ConjPoint}(\mathbf{M}_{3j+1}, \varphi_p) = l'''$, where $p \in S_3$, then

Let w_3 be a member of S_3. Let $p'''_{3j+1} = \mathbf{M}_{3j+1}(\varphi_{w_3}[l'''])$. Let Z'' denote $Z(p_1, \ldots, p_{2j}, p'_{2j+1}, \ldots, p'_{3j}, p'''_{3j+1}, \varphi_{w_3}[l'''])$. Let $\varphi_{W''} = \varphi_{Z''} = f_2$, where f_2 is as in $\text{Simul}(\{p_1, \ldots, p_{2j}, p'_{2j+1}, \ldots, p'_{3j}, p'''_{3j+1}\}, j + 1, \varphi_{w_2}[l'''], m_2, f_1, f_2)$.

end $\varphi_{X_i(p_1, p_2, \ldots, p_{2j}, f[n])}, i = 1, 2$

Suppose f is $\mathbf{Team}^{2j}_{4j}\mathbf{Fin}$-identified by $\mathbf{M}_1, \ldots, \mathbf{M}_{4j}$. Let the programs output (if any) by machines $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_{4j}$ on f be $p_1, p_2, \ldots, p_{4j}$ respectively. For ease of notation let X_i denote $X_i(p_1, p_2, \ldots, p_{2j}, f[n])$, where $n = \text{ConjPoint}(\mathbf{M}_{2j}, f)$. Let Y denote $Y(p_1, p_2, \ldots, p_{3j}, f[m])$, where $m = \text{ConjPoint}(\mathbf{M}_{3j}, f)$. Let Z denote $Z(p_1, p_2, \ldots, p_{3j+1}, f[l])$, where $l = \text{ConjPoint}(\mathbf{M}_{3j+1}, f)$. We now consider the following cases.

Case 1: $\mathbf{M}_{3j}$ does not output a program on f.

In this case, there are at least $j + 1$ correct programs (for f) in $\{p_1, \ldots, p_{2j}\}$ and on f since $\mathbf{M}_{3j}$ does not output a program, step 1.1. in the procedure for φ_{X_i} will not succeed, and thus both X_1, X_2 will simulate f.

Case 2: $\mathbf{M}_{3j}$ outputs a program on f, but $\mathbf{M}_{3j+1}$ does not output a program on f.

In this case, in the procedure for φ_{X_i} step 1.1. will eventually succeed. Also in the procedure for φ_Y step 1.1. eventually succeeds (since step 1.2. can never succeed). If $f[m]$ is not contained in φ_{X_1}, then consider step 3 onwards in the computation of φ_{X_2} (see description in step 2.2). Otherwise consider step 3 onwards in the computation of φ_{X_1}. Note

that at least $2j$ of the programs in $\{p_1, \ldots, p_{3j}\}$ are program for f, and thus steps 5.1. or 5.2. cannot succeed. Thus either due to step 3.2 to 3.5 or due to steps 5.3 to 5.6, at least two of $\varphi_{X_1}, \varphi_{X_2}, \varphi_Y$ are same as f.

Case 3: $\mathbf{M}_{3j+1}$ outputs a program on f.

Case 3.1: Less than j of the programs in $\{p_1, p_2, \ldots, p_{2j}\}$, calculate $f[m]$ correctly.

In this case, in the procedure for φ_Y step 1.2. would eventually succeed (and step 1.1. can never succeed) and thus both Y, Z are programs for f.

Case 3.2: Greater than j of the programs in $\{p_1, p_2, \ldots, p_{2j}\}$, calculate $f[m]$ correctly.

In this case, note that the procedure for φ_{X_i} reaches step 2, and the condition for splitting off φ_{X_2} stated in step 2.2. never succeeds.

Case 3.2.1: There are at least $2j+1$ programs in the $\{p_1, \ldots, p_{3j}\}$ which compute upto $f[l]$ correctly.

In this case less than j programs in $\{p_1, \ldots, p_{3j}\}$ are inconsistent with $f[l]$.

Thus we have that f_2 in Simul$(\{p_1, \ldots, p_{3j+1}\}, j+1, f[l], m, f_1, f_2)$, is simulated by Y, Z (either in step 3 of procedure for φ_Y, or step 4 of procedure for φ_{X_i}) and f_1 is simulated by X_1, X_2 at step 6.1. of φ_{X_i}.

Case 3.2.2: There are less than $2j+1$ programs in the $\{p_1, \ldots, p_{3j}\}$, which compute upto $f[l]$ correctly.

Note that if there are at least j programs in $\{p_1, \ldots, p_{3j}\}$, which extend some $g[k] \supseteq f[m]$, on which $\mathbf{M}_{3j+1}$ outputs a program, then corresponding Z (as output by $\mathbf{M}'_4$ on function g) and at least one of X_1, X_2, Y, simulate f_2 in Simul$(\{p_1, \ldots, p_{3j+1}\}, j+1, g[k], m, f_1, f_2)$. Since for f there could be at most $2j+1$ programs in $\{p_1, \ldots, p_{3j+1}\}$ which extend $f[l]$, we have that Z and one of Y, X_1, X_2 compute f (by properties of Simul as described in Lemma 1).

Case 3.3: There are exactly j programs in $\{p_1, \ldots, p_{2j}\}$, which calculate $f[m]$ correctly.

In this case in procedure for φ_{X_i} step 1.1. eventually succeeds. Suppose, without loss of generality that X_1 computed $f[m]$ correctly (case for X_2 is similar). In this case Z, Y simulate f_2 in Simul$(\{p_1, \ldots, p_{3j+1}\}, j+1, f[l], m, f_1, f_2)$, at either step 3 of φ_Y or step 4 of φ_{X_1}. They thus compute f. ∎

4 Conclusions

In this paper we presented some results about finite identification of programs from graphs of computable functions by teams of deterministic machines, and contrasted some of these results with corresponding results about limiting team function inference. As a consequence of this study a direct analog of Pitt's connection does not hold for finite identification. Also the results presented here along with the results in [DPVW91] complete the picture for finite identification for ratio $\geq 1/2$. Recently there has been significant work for success ratios $< \frac{1}{2}$ and related issues. For example in [DPVW91], it is shown that at success ratio 1/3, introducing redundancy always helps. We direct the reader to [DKV92a, DK93, DKV93, DKV92b]. However, the picture is far from complete.

For related work on language identification by teams we direct the reader to [JS90b, JS93c, JS93a, JS93b].

5 Acknowledgements

We would like to thank John Case, Mark Fulk, Bill Gasarch, Sudhir Jha, Lata Narayanan, and Rajeev Raman for helpful discussion and comments. We are also grateful to the two referees for providing several helpful comments. Preliminary version of results described in this paper were presented at the *Second Annual Workshop on Computational Learning Theory* [Vel89] and the *Third Annual Workshop on Computational Learning Theory* [JS90a].

References

[BB75] L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.

[Blu67] M. Blum. A machine independent theory of the complexity of recursive functions. *Journal of the ACM*, 14:322–336, 1967.

[CS83] J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.

[DK93] R. P. Daley and B. Kalyanasundaram. Capabilities of probabilistic learners with bounded mind changes. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory, Santa Cruz, California*, pages 182–191. A. C. M. Press, 1993.

[DKV92a] R. P. Daley, B. Kalyanasundaram, and M. Velauthapillai. Breaking the probability 1/2 barrier in fin-type learning. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, Pittsburgh, Pennsylvania*, pages 203–217. A. C. M. Press, 1992.

[DKV92b] R. P. Daley, B. Kalyanasundaram, and M. Velauthapillai. The power of probabilism in popperian finite learning. In *Proceedings of the Third International Workshop on Analogical and Inductive Inference, Dagstuhl Castle, Germany*, pages 151–169, October 1992.

[DKV93] R. P. Daley, B. Kalyanasundaram, and M. Velauthapillai. Capabilities of fallible finite learning. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory, Santa Cruz, California*, pages 199–208. A. C. M. Press, 1993.

[DPVW91] R. P. Daley, L. Pitt, M. Velauthapillai, and T. Will. Relations between probabilistic and team one-shot learners. In L. Valiant and M. Warmuth, editors, *Proceedings of the Workshop on Computational Learning Theory*, pages 228–239. Morgan Kaufmann Publishers, Inc., 1991.

[Fre79] R. Freivalds. Finite identification of general recursive functions by probabilistic strategies. In *Proceedings of the Conference on Fundamentals of Computation Theory*, pages 138–145. Akademie-Verlag, Berlin, 1979.

[Gil77] Gill. Computational complexity of probabilistic turing machines. *SIAM Journal of Computing*, 1977.

[Gol67] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[JS90a] S. Jain and A. Sharma. Finite learning by a team. In M. Fulk and J. Case, editors, *Proceedings of the Third Annual Workshop on Computational Learning Theory, Rochester, New York*, pages 163–177. Morgan Kaufmann Publishers, Inc., August 1990.

[JS90b] S. Jain and A. Sharma. Language learning by a team. In M. S. Paterson, editor, *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, pages 153–166. Springer-Verlag, July 1990. Lecture Notes in Computer Science, 443.

[JS93a] S. Jain and A. Sharma. Computational limits on team identification of languages. Technical Report 9301, School of Computer Science and Engineering; University of New South Wales, 1993.

[JS93b] S. Jain and A. Sharma. On aggregating teams of learning machines. In K.P. Jantke, S. Kobayashi, E. Tomita, and T. Yokomori, editors, *Proceedings of the Fourth International Workshop on Algorithmic Learning Theory, Tokyo, Japan*, pages 150–163. Springer-Verlag, November 1993. Lecture Notes in Artificial Intelligence No. 744.

[JS93c] S. Jain and A. Sharma. Probability is more powerful than team for language identification. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory, Santa Cruz, California*, pages 192–198. ACM Press, July 1993.

[MY78] M. Machtey and P. Young. *An Introduction to the General Theory of Algorithms*. North Holland, New York, 1978.

[OSW86] D. Osherson, M. Stob, and S. Weinstein. Aggregating inductive expertise. *Information and Control*, 70:69–95, 1986.

[Pit89] L. Pitt. Probabilistic inductive inference. *Journal of the ACM*, 36:383–433, 1989.

[PS88] L. Pitt and C. Smith. Probability and plurality for aggregations of learning machines. *Information and Computation*, 77:77–92, 1988.

[Rog67] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw Hill, New York, 1967. Reprinted, MIT Press 1987.

[Smi82] C. Smith. The power of pluralism for automatic program synthesis. *Journal of the ACM*, 29:1144–1165, 1982.

[Smu61] R. Smullyan. *Theory of Formal Systems, Annals of Mathematical Studies, No. 47*. Princeton, NJ, 1961.

[TB70] B. Trakhtenbrot and J. M. Barzdin. *Konetschnyje Awtomaty (Powedenie i Sintez) (in Russian)*. Nauka, Moskwa, 1970. English Translation: Finite Automata–Behavior and Synthesis, Fundamental Studies in Computer Science 1, North Holland, Amsterdam, 1975.

[Vel89] M. Velauthapillai. Inductive inference with bounded number of mind changes. In *Proceedings of the Second Annual Workshop on Computational Learning Theory, Santa Cruz, California*, pages 200–213. Morgan Kaufmann Publishers, Inc., August 1989.