Fundamenta Informaticae 30 (1997) 23–41
IOS Press

Asking Questions versus Verifiability[*]

William Gasarch
Department of Computer Science
University of Maryland at College Park
College Park, MD, 20742, USA
gasarch@cs.umd.edu

Mahendran Velauthapillai
Department of Computer Science
Georgetown University
Washington, DC, 20057, USA
mahe@cs.georgetown.edu

Keywords: Inductive inference, queries.

1. Introduction

Inductive inference is the study of learning in a recursion-theoretic framework. The basic model is that the learner sees data $f(0), f(1), \ldots$, makes conjectures $e_1, e_2, \ldots$ (these are programs) and eventually always outputs the same program, which is a program for f. This model was introduced by Gold [12]. This model, and several variants of it, have been extensively studied by several people including [2, 3, 16]. See [1, 10, 8] for several surveys.

Case and Smith [3] studied learning machines whose conjectures are verifiable (i.e. the conjectures are total programs). They discovered that such machines are *weaker* then machines that are allowed to conjecture non-total programs (in symbols $PEX \subset EX$).

Gasarch and Smith [9] studied machines that learn functions by asking questions about the function (see [6, 7] for subsequent work). They discovered that machines that can ask questions can usually learn *more* than machines that do not (in symbols $EX \subset Q_1EX[*]$).

These two adjustments on the original model, and the results about them, lead to the following question: *Can the weakness of verifiability be overcome by the strength of asking questions?* While studying this question several other issues arise, including a more detailed study of verifiability.

2. Definitions

Throughout this paper, φ_0, φ_1, φ_2, $\ldots$ denotes an *acceptable programming system* [17], also known as a *Gödel numbering of the partial recursive functions* [15]. The function φ_e is said to be computed by the program e.

[*]First author supported in part by NSF grants CCR-9020079 and CCR-9301339

An (standard, passive) *inductive inference machine* (IIM) is a total algorithmic device that takes as input the graph of a recursive function (an ordered pair at a time) and outputs (from time to time) programs intended to compute the function whose graph serves as input [2, 3, 12]. An IIM M *learns a recursive function* f, if, when M is given the graph of f as input, the resultant sequence of outputs converges (after some point there are no more mind changes) to a program that computes f. This is denoted by $M(f)\downarrow$. Learning has occurred since we can use the program output by the IIM to predict the value of $f(x)$, for all x, after having seen only finitely many examples. In this case we write $f \in EX(M)$. (EX stands for "explains" [3] since we think of the program as being an explanation of the data.) The class EX is the collection of all sets $EX(M)$ (or subsets thereof) of functions learned by an IIM. We say that $f \in PEX(M)$ if the IIM only outputs programs which compute *total* recursive functions, and eventually converges to a single program which computes f. This enables one to verify that the programs output by the IIM agrees with the data seen so far and with later data as it comes in. The collection of all the sets $PEX(M)$ is called PEX. (See [3] for why it is called PEX.) If σ is an initial segment of some function f, then $M(\sigma)$ denotes the last conjecture about f made by M, after seeing all of σ as input, but prior to requesting additional input.

Each time an IIM outputs a conjecture we say that another *learning trial* has been completed. Since it is never known if enough trials have been completed, it is sometimes desirable to fix an a priori bound on the number of trials that will be permitted. If convergence is achieved after only c changes of conjecture we write $f \in EX_c(M)$, ($f \in PEX_c(M)$) for $c \in \mathsf{N}$, where N denotes the set of natural numbers. The class of sets of functions identifiable by IIMs restricted to c mind changes is denoted by EX_c (PEX_c).

The requirement that M on input f converges to a program φ_e such that φ_e computes f on *every* point is somewhat stringent. One relaxation of this to allow the the machine to converge to a program that computes f on most points, but we will allow a (or less) exceptions. $f \in EX_b^a(M)$ if and only if M on some initial segment of f will converge a program e after at most b mind changes, and $|\{x : \varphi_e(x) \neq f(x)\}| \leq a$. The notions of EX^a, PEX^a and PEX_b^a can be defined similarly. For $PEX_b^a(M)$ we require the errors to be errors of commission.

Smith [16] motivated and defined the following notion of *team inference*. Let $M_1, \ldots, M_d$ be a set of d IIMs. Let f be a recursive function. If at least c of $\{M_1, \ldots, M_d\}$ correctly EX-infer f then f *is* $[c,d]$*-inferred by* $\{M_1, \ldots, M_d\}$. $S \in [c,d]EX$ if there exists $M_1, \ldots, M_d$ such that, for every $f \in S$, f is $[c,d]$*-inferred by* $\{M_1, \ldots, M_d\}$. The set of machines $\{M_1, \ldots, M_d\}$ is referred to as a *team*. Teams of any type of inference machine can be defined similarly. We will be using $[c,d]PEX_a^b$.

A *query inference machine* (QIM), defined by Gasarch and Smith [9], is an algorithmic device that asks a teacher questions about some unknown function and, while doing so, outputs programs. In this way the QIM learns a recursive function by asking questions about that function. We assume that the teacher always returns the correct answer to any question asked by a QIM. The questions are formulated in some query language L. A variety of different query languages are considered. The languages that we consider have different expressive power. The more expressive the query language, the more types of questions the QIM can ask. In a sense, giving a QIM a more expressive query language to use makes the QIM more articulate. Formally, a QIM is a total algorithmic device which, if the input is a string of bits $\vec{b}$, corresponding to the answers to previous queries, outputs an ordered pair consisting of a guess which is a (possibly null) program e, and a question ψ. For more details and technical results about QIM's see [6, 7, 9].

Define two functions g (*guess*) and q (*query*) such that if $M(\vec{b}) = (e, \psi)$ then $g(M(\vec{b})) = e$ and $q(M(\vec{b})) = \psi$. By convention, all questions are assumed to be sentences in prenex normal form (quantifiers followed by a quantifier-free formula, called the matrix of the formula) and questions containing quantifiers are assumed to begin with an existential quantifier. This convention entails no loss of generality and serves to reduce the number of cases needed in several proofs. A QIM M *learns a recursive function* f if, when the teacher answers M's questions about f truthfully, the sequence of output programs (total programs) converges to a program that computes f. In this case, we write $f \in QEX[L](M)$ ($f \in QPEX[L](M)$). For a fixed language L, the class $QEX[L]$ ($QPEX[L]$) is the collection of all sets $QEX[L](M)$ ($QPEX[L](M)$) as M varies over all QIMs that use the query language L. Note in the case of $QPEX[L](M)$, even if the QIM M is given the wrong answers, it has to output total programs. We choose this definition of $QPEX$ because we want to make it analogous to PEX: with both PEX and $QPEX$ even with bad information, the conjecture must be total.

All the query languages that we will consider allow the use of quantifiers. Restricting the applications of quantifiers is a technique that we will use to regulate the expressive power of a language. Of concern to us is the alternations between blocks of existential and universal quantifiers. Suppose that $f \in QEX[L](M)$ ($f \in QPEX[L](M)$) for some M and L. If M only asks quantifier-free questions, then we will say that $f \in Q_0EX[L](M)$ ($f \in Q_0PEX[L](M)$). If M only asks questions with existential quantifiers, then we will say that $f \in Q_1EX[L](M)$ ($f \in Q_1PEX[L](M)$). In general, if M's questions begin with an existential quantifier and involve $a > 0$ alternations between blocks of universal and existential quantifiers, then we say that $f \in Q_{a+1}EX[L](M)$ ($f \in Q_{a+1}PEX[L](M)$). The classes $Q_aEX[L]$, $Q_aPEX[L]$, $Q_aEX_c[L]$ and $Q_aPEX_c[L]$ are defined analogously. By convention, if a QIM restricted to c mind changes actually achieves that bound, then it will ask no further questions.

Now we introduce the query languages that will be used. Every language allows the use of $\wedge$, $\neg$, $=$, $\forall$, $\exists$, symbols for the natural numbers (members of N), variables that range over N, and a single unary function symbol $\mathcal{F}$ which will be used to represent the function being learned. Inclusion of these symbols in every language will be implicit. The *base* language contains only these symbols. If L has auxiliary symbols, then L is denoted just by these symbols. For example, the language that has auxiliary symbols for plus and less than is denoted by $[+, <]$. The language that has auxiliary symbols for plus and times is denoted by $[+, \times]$. The language with extra symbols for successor and less than is denoted by $[S, <]$, where S indicates the symbol for the successor operation.

We will often want to state that a theorem is true for any language one might consider.

Definition 2..1 A language L is *reasonable* if all the symbols in it denote recursive operations.

Notation 2..2 Let $\mathcal{A}$ be some inference class. We define the notation $QEX[\star]$ in terms of how it is used. The analogous definitions for $QPEX$ and other variants of QEX can easily be defined.

1. $QEX[\star] \subseteq \mathcal{A}$ means that if L is a reasonable language then $QEX[L] \subseteq \mathcal{A}$.
2. $\mathcal{A} \subseteq QEX[\star]$ means that for any L (including $L = \emptyset$) $\mathcal{A} \subseteq QEX[L]$.
3. $QEX[\star] - \mathcal{A} \neq \emptyset$ means that for any L (including $L = \emptyset$) $QEX[L] - \mathcal{A} \neq \emptyset$ means
4. $\mathcal{A} - QEX[\star] \neq \emptyset$ means that for any reasonable L. $\mathcal{A} - QEX[L] \neq \emptyset$ means

The symbol "$\star$" will be used to denote an arbitrary language that includes all the symbols common to all the languages we consider and some (possibly empty) subset of operational symbols denoting computable operations.

3. Technical Summary

Before comparing PEX to $QPEX$ it is necessary to examine PEX in more detail. In Section 4. we look at the four parameter problem for PEX, namely, 'for what values of $a, b, c,$

an d does $PEX_b^a \subseteq PEX_c^d$?' It turns out that, unlike the case for EX, there are interesting tradeoffs. In Section 5. we look at how PEX compares to $QPEX[*]$. We show that $QPEX[*] = PEX$, which answers the motivating question of this paper: The strength of asking questions *does not* make up for the weakness of outputting verifiable conjectures. In Section 6. we reexamine this question with bounded mind changes. We show that $QPEX_0[*] - PEX_a \neq \emptyset$. In Sections 7. and 8. we look at the particular languages $[+,\times]$, $[+,<]$, and $[S,<]$.

4. PEX Learning

The first result shows the mindchange-anomaly trade-offs for PEX machines. The next two results extends this to certain teams. The moral of the story will be that mindchanges are the most important parameter. The theorems proved in this section uses techniques from [3, 16].

Theorem 4.1. Let $a, b, c, d \in \mathsf{N}$. Then

$$PEX_b^a \subseteq PEX_d^c \text{ if and only if } d+1 \geq \Big(b+1\Big)\Big(\Big\lfloor\frac{a}{c+1}\Big\rfloor + 1\Big).$$

Proof:

($\Longleftarrow$) Let $a, b, c, d \in \mathsf{N}$ such that

$$d+1 \geq \Big(b+1\Big)\Big(\Big\lfloor\frac{a}{c+1}\Big\rfloor + 1\Big).$$

Let $S \in PEX_b^a$ via M. We will construct an IIM M' such that $S \in PEX_d^c$ via M'. M' operates as follows: keep outputing a guess that M output until either $c+1$ anomalies are spotted (this can be tested since M only outputs total functions) or M outputs a new guess. In the former case M' outputs a patched version of the program that corrects the $c+1$ anomalies (unless $a+1$ anomalies to this program have already been spotted in which case output the same program that was output before). In the latter case merely output the new index.

For each guess that M outputs M' will output at most $\lfloor\frac{a}{c+1}\rfloor + 1$ guesses. Since M outputs at most $b+1$ guesses we have that M' outputs at most $(b+1)(\lfloor\frac{a}{c+1}\rfloor + 1)$ guesses. Hence M' makes at most $(b+1)(\lfloor\frac{a}{c+1}\rfloor + 1) - 1 \leq d$ mindchanges. The final guess must be correct on all but at most c places since $c+1$ anomalies would have caused a new program with those points patched up to be output.

($\Longrightarrow$)

We show that if $d < \Big(b+1\Big)\Big(\lfloor\frac{a}{c+1}\rfloor + 1\Big) - 1$ then $PEX_b^a - PEX_d^c \neq \emptyset$. Let $\mathcal{B}_s = \{\{b+1, b+2\}^{c+1} \cup s^*\}^{\lfloor\frac{a}{c+1}\rfloor}$. Let S be the union of the following sets.

$$\begin{array}{l}
\{0^{\geq a+1}\mathcal{B}_0 0^\omega\} \\
\{0^{\geq a+1}\mathcal{B}_0 1^{\geq a+1}\mathcal{B}_1 1^\omega\} \\
\{0^{\geq a+1}\mathcal{B}_0 1^{\geq a+1}\mathcal{B}_1 2^{\geq a+1}\mathcal{B}_2 2^\omega\} \\
\{0^{\geq a+1}\mathcal{B}_0 1^{\geq a+1}\mathcal{B}_1 2^{\geq a+1}\mathcal{B}_2 3^{\geq a+1}\mathcal{B}_3 3^\omega\} \\
\vdots \\
\{0^{\geq a+1}\mathcal{B}_0 1^{\geq a+1}\mathcal{B}_1 2^{\geq a+1}\mathcal{B}_2 \cdots \mathcal{B}_{b-2}(b-1)^{\geq a+1}\mathcal{B}_{b-1} b^{\geq a+1}\mathcal{B}_b b^\omega\}.
\end{array}$$

The following algorithm shows that $S \in PEX_b^a$.

1) Look for s^{a+1} where $s \in \{0, \ldots, b\}$ and s has not been seen in the range of f yet.
2) Output an index for a program which has an initial segment of all the values M has seen so far and the rest with the constant value seen in step (1).
3) Go to step (1).

The inference procedure clearly changes its mind at most b times. Assume that the final index output thinks the function is almost all s. Hence no $s+1$'s are ever seen. The number of errors possible are just those that occur in the $\mathcal{B}_s$ that are not s. This is at most $(c+1)(\left\lfloor \frac{a}{c+1} \right\rfloor) \le a$. Hence this procedure shows $S \in PEX_b^a$.

We show $S \notin PEX_c^d$. Assume, by way of contradiction, that $S \in PEX_c^d$ via M'. We will construct an $f \in S$ such that $f \notin PEX_d^c(M')$. The construction of f will proceed in stages. At stage s f^s will denote the initial segment of f that has been constructed so far. We will use s to indicate the values by which the functions is extended. Initially set $s = 0$ and $\sigma = \lambda$

For $s = 0$ *to* b *do begin*

1) Extend σ with at least $a+1$ values of s until M' on σ outputs a new guess. (This has to happen since we will see later that $\sigma s^\omega \in S$.)
2) Let $M'(\sigma) = e$. Extend σ to $\sigma\tau$ where $\tau \in \{b+1, b+2\}^{c+1}$ and φ_e is wrong on every point of τ. This can be done since φ_e is total. This extension makes the current guess wrong on an additional $\ge c+1$ places.
3) Extend by s's until a new guess is made. this must happen since the current guess is wrong on $c+1$ places.
4) Repeat steps 2 and 3 $\left\lfloor \frac{a}{c+1} \right\rfloor - 1$ times.

It is easy to show inductively that after stage s of the construction we have

$$\sigma \in \{0^{\ge a+1}\mathcal{B}_0 1^{\ge a+1}\mathcal{B}_1 2^{\ge a+1}\mathcal{B} \cdots s^{\ge a+1}\mathcal{B}_s\}.$$

Hence $\sigma s^\omega \in S$ so step 1 of the construction always terminates. Hence we succeed in constructing f^b. We now estimate how many mindchanges occur when we feed f^b into M. Every iteration of the loop yields $\left\lfloor \frac{a}{c+1} \right\rfloor + 1$ guesses. There are $b+1$ iterations of the loop, so $(b+1)(\left\lfloor \frac{a}{c+1} \right\rfloor + 1)$ guesses are made, so there are at most $(b+1)(\left\lfloor \frac{a}{c+1} \right\rfloor + 1) - 1 > d$ mindchanges. Hence M cannot PEX_d^c-infer S.

Theorem 4.2. Let $a, b \in \mathsf{N}$ and $n \ge 1$. Then $[1,n]PEX_b^a = PEX_{n(b+1)-1}^a$.

Proof:

Let $S \in PEX_{n(b+1)-1}^a$ via M. Note that M makes at most $n(b+1)$ guesses on each function it infers. We exhibit machines $M_1, \ldots, M_n$ such that $S \in [1,n]PEX_b^a$ via $M_1, \ldots, M_n$. M_i behaves as follows: when M outputs its jth guess, $(i-1)b + i \le j \le ib + i$, M_i outputs that guess. Note that M_n outputs guesses from $(n-1)b + n$ to $nb + n = n(b+1)$. Clearly each M_i makes at most $b+1$ guesses and one of them uses the last guess (hence the correct guess) made by M. This last guess will differ from the function to be inferred by at most b.

Let $S \in [1,n]PEX_b^a$ via $M_1, \ldots, M_n$. We exhibit a program M' such that $S \in PEX_{n(b+1)-1}^a$ via M'.

1) Output the lexicographically least program output by the by the team (which has not been previously output by M').
2) Run the program output in step (1) on data as it comes in. If $(a+1)$ errors are found then go to step (1).

END M'

The maximum number of programs that can be output is clearly $n(b+1)$. Hence M' changes its mind at most $n(b+1) - 1$ times.

Theorem 4.3. Let $a, b, c, d, n, m \in \mathsf{N}$. Then

$$[1, n]PEX^a_b \subseteq [1, m]PEX^c_d \text{ if and only if } m(d+1) \geq n\Big(b+1\Big)\Big(\Big\lfloor \frac{a}{c+1} \Big\rfloor + 1\Big).$$

Proof:
This is obtained by combining Theorems 4.1. and 4.2..

5. QPEX Learning with Any Language

In [9] we saw that the ability to ask questions *did* increase the set of classes learnable. In that paper the machines were allowed to output non-total programs. We now consider what happens if the machine must output total programs. In the case of unbounded mindchanges *there is no increase in power.*

Theorem 5.1. $QPEX[\star] = PEX$.

Proof:
Clearly $PEX \subseteq QPEX[\star]$. Now we will show that $QPEX[\star] \subseteq PEX$. Let $S \in QPEX[\star]$. Then there exists a QIM M such that $S \subseteq QPEX[\star](M)$. Now we will construct an IIM M' such that $S \in PEX(M)$. The proof of the theorem is based on the fact that if QIM M infers any $f \in S$ then there exists a sequence of answers which will eventually lead to the correct program. M' executes the following algorithm for any $f \in S$.

```
0) Let $BITSTRINGS = \{0,1\}^*$.
1) Let $\vec{b}$ be the lexicographically least string in $BITSTRINGS$.
2) Let $e = g(M(\vec{b}))$, output $e$. Note that $e$ is a total program.
3) Check more and more values of $f$ with $\varphi_e$, until an error is found (this may never
   happen).
4) If an error is found in step (3) delete $\vec{b}$ from $BITSTRINGS$ and go to step (1).
```

Let $\vec{b}$ be the least (lexicographically) element of $\{0,1\}^\star$ that leads to a program that computes f. Such exists since the sequence of correct answers leads to a program that computes f. M' will eventually find $\vec{b}$ and use if forever more since all strings $\vec{c}$ that are less than $\vec{b}$ lead to incorrect programs and will hence be eliminated. Once $\vec{b}$ is being used, M' will output the correct program forever more.

We now show that part of Theorem 4.1. and all of Theorem 4.2. hold for $QPEX$ as well as PEX.

Theorem 5.2. Let $a, b, c, d \in \mathsf{N}$. If

$$d + 1 \geq \Big(b+1\Big)\Big(\Big\lfloor \frac{a}{c+1} \Big\rfloor + 1\Big)$$

then $QPEX^a_b \subseteq QPEX^c_d$.

Proof:
The proof of this theorem is similar to the proof of the first part of Theorem 4.1..

Theorem 5.3. Let $a, b \in \mathsf{N}$ and $n \geq 1$. Then $[1, n]QPEX^a_b = QPEX^a_{n(b+1)-1}$.

Proof:
The proof of this theorem is similar to the proof of the Theorem 4.2..

6. Bounded Mindchanges

In Theorem 5.1. we proved that when unbounded mind changes are allowed the ability to ask questions does not increase power. However in the next theorem we will show that if the mind changes are bounded the ability to ask questions with just one quantifier will increase power. To state the theorem we need one more definition.

Definition 6..1 $EX^\star$ is similar to EX except that we allow the final program to have a finite number of errors. One can define such a notion for PEX and all other inference classes as well.

Theorem 6.1. $(\forall a \in \mathsf{N})[Q_1PEX_0[\star] - PEX_a^\star \neq \emptyset.]$

Proof:
Let
$$S = \bigcup_{i=0}^{a} \{0^{n_0}1^{n_1}\cdots i^{n_i}(i+1)^\omega : n_0, \ldots, n_i \in \mathsf{N}\}.$$

The following QIM algorithm shows $S \in Q_1PEX_0[\star]$.

1) Ask $(\exists x)[f(x) = a+1]$. If the answer is NO then ask $(\exists x)[f(x) = a]$. Continue these questions for less and less values until the answer is YES for some constant c, $c \leq a+1$.
2) Find the value of $f(0)$ by asking questions of the form "is $f(0) = 0$?", $f(0) = 1$? and so on. Similarly find the values of $f(1)$,$f(2)$,$\cdots$ $f(x-1)$. Stop when a b is found such that $f(b) = c$.
3) Output an index for the function $f(0)f(1)\cdots f(b)c^\omega$.

We show that $S \notin PEX_a^\star$. Let M be any IIM. We construct $f \in S$ such that f is not in $PEX_a^\star$ via M. The construction of f will proceed in stages. f^s denotes the initial segment of f constructed at stage s. Set $f^0 = \lambda$. Extend f^0 with more and more values of zero until M outputs a program on f^0.

For $s = 0$ **to** a **do**

1) Look for $i \in \mathsf{N}$ and $t \in \{s, s+1\}$ such that $M(f^st^i) \neq M(f^s)$. This must happen since both f^ss^ω and $f^s(s+1)^\omega$ are in S, and the current index cannot be $*$-correct for both of them.
2) Set f^{s+1} to f^st^i.

Clearly the loop must terminate. Let the maximum domain value at which f^{a+1} is defined be b. Now set $f = f^a \cdot b^\omega$ Clearly the IIM M will change its mind $a+1$ times on f. It is only allowed a mind changes

Theorem 6.2. $(\forall a, b, c \in \mathsf{N} \cup \{\star\})$ $[QPEX_b^a[\star] \subseteq PEX_c$ if and only if $c = \star]$.

Proof:
$(\Rightarrow)$ Let $(\forall a, b, c \in \mathsf{N} \cup \{\star\})$. Suppose $c \neq \star$, then by Theorem 6.1. $QPEX_b^a[\star] - PEX_c \neq \emptyset$. Hence if $QPEX_b^a[\star] \subseteq PEX_c$ implies that $c = \star$. $(\Leftarrow)$ Let $c = \star$, then by Theorem 5.1. the result follows.

7. PEX Learning with $[+, \times]$

In this section we will show that if $L = [+, \times]$ then going from Q_0PEX_0 to Q_1PEX_0 causes a large increase in learning power.

Theorem 7.1. $Q_0PEX_0[+, \times] = PEX_0$.

Proof:
By definitions.

Theorem 7.2. $Q_1PEX_0[+, \times] = PEX$.

Proof:
By Theorem 5.1. $Q_1PEX_0[+, \times] \subseteq PEX$. We show the reverse inclusion.

Let $S \in PEX$ via M. We show $S \in Q_1PEX_0[+, \times]$. Let $\langle ., . \rangle$ denote a recursive pairing function from $\mathsf{N} \times \mathsf{N} \rightarrow \mathsf{N}$. For $e \in \mathsf{N}$ let $A_e = \{\langle x, y \rangle : \varphi_e(x) \downarrow\neq y\}$. Clearly A_e is r.e. By work done in solving Hilberts 10th problem [4, 14] (also Theorem 8 in [9]) there is an effective list of polynomials p_0,p_1,$\cdots$ such that for all i,

$$\langle x, y \rangle \in A_i \Leftrightarrow \exists \vec{z}[p_i(\vec{z}, \langle x, y \rangle) = 0].$$

That is

$$\varphi_i(x) \downarrow\neq y \Leftrightarrow \exists \vec{z}[p_i(\vec{z}, \langle x, y \rangle) = 0].$$

We $Q_1PEX_0[+, \times]$-infer S as follows. Let $f \in S$. Finds the values of $f(0)$, $f(1)$,$f(2)$,$\cdots$ and feeds them into M. When M outputs a guess e, we ask the following question.

$$\exists x \exists \vec{z}[p_{e_1}(\vec{z}, \langle x, f(x) \rangle) = 0]$$

Note that the above question is equivalent to $\exists x[\varphi_e(x) \downarrow \not= f(x)]$.

If the answer is NO, then output e and stop. Note that *since* φ_e *is total* an answer of NO means that φ_e computes f. If the answer is YES then it waits for a new guess and repeats the procedure. Eventually, when the correct index is output by M, we will receive an answer of NO and output this correct index.

8. Learning with $[+, <]$ and $[S, <]$

If $L = [+, <]$ or $L = [S, <]$, and mindchanges are bounded, then $QPEX_a[L]$ is strictly weaker than PEX. The following theorem formalizes this. It was originally proven in an earlier version of this paper [11], using k-good sets (from [7]). There is now a new proof using ω-automata that is much simpler; it appears in [5].

Theorem 8.1. If $L = [+, <]$ or $L = [S, <]$ and $a \in \mathsf{N}$ then $PEX_a - QPEX_{a-1}[L] \neq \emptyset$.

9. Open Problems

We have determined (in Theorem 4.3.) exactly when $[1, n]PEX_a^b \subseteq [1, m]PEX_c^d$. The general problem of determining exactly when $[e, f]PEX_a^b \subseteq [g, h]PEX_c^d$ is open. The case of $[e, f]PEX_0 \subseteq [g, h]PEX_0$ has been solved by Kummer [13] and independently by Kalyanasundaram. It is quite complicated. That is, there is no formula for the condition, just a recursive algorithm to determine when the condition holds.

Let L be some language. We have a condition that implies $QPEX_a^b[L] \subseteq QPEX_c^d[L]$ (Theorem 5.2.). Is the converse true? More generally, it is open to determine exactly when $QPEX_a^b[L] \subseteq QPEX_c^d[L]$. By Theorem 5.3. the solution to this problem will yield a solution to the problem of when $[1,n]QPEX_a^b[L] \subseteq [1,m]QPEX_c^d[L]$. The general problem of determining when $[e,f]QPEX_a^b \subseteq [g,h]QPEX_c^d$ is open.

Let $[S,<,2]$ denote the query language where we allow quantification over sets (second order quantification, hence the '2'). There is a connection between queries in $[S,<,2]$ and ω-automata which has made it possible to prove theorems about $[S,<,2]$ [9, 5]. Once results about $[S,<,2]$ are obtained it is often possible to use reduction techniques [5] to obtain results about many other languages. This may help resolve some of our open problems.

References

[1] D. Angluin and C. H. Smith. A survey of inductive inference: Theory and methods. *ACM Comput. Surv.*, 15(3):237–269, Sept. 1983.

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

[3] J. Case and C. H. Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Comput. Sci.*, 25:193–220, 1983.

[4] M. Davis, H. Putnam, and J. Robinson. The decision problem for exponential diophantine equations. *Annals of Mathematics*, 74:425–436, 1961.

[5] W. Gasarch and G. Hird. Automata techniques for query inference machines. *Annals of pure and applied logic.* to appear. Earlier version in COLT 1995.

[6] W. Gasarch, E. Kinber, M. Pleszkoch, C. H. Smith, and T. Zeugmann. Learning via queries, teams, and anomalies. *Fundamenta Informaticae*, 23:67–89, 1995. Shorter version in Third Annu. Conference on Computational Learning Theory, 1990, pages 327-337, published by Morgan Kaufman.

[7] W. Gasarch, M. Pleszkoch, and R. Solovay. Learning via queries to $[+,<]$. *Journal of Symbolic Logic*, 57(1):53–81, Mar. 1992.

[8] W. Gasarch and C. Smith. A survey of inductive inference with an emphasis on learning via queries. In A. Sorbi, editor, *Complexity, Logic, and Recursion Theory.* M. Dekker, 1997.

[9] W. Gasarch and C. H. Smith. Learning via queries. *Journal of the ACM*, 39(3):649–675, July 1992. A shorter version is in 29th FOCS conference, 1988, pp. 130-137.

[10] W. Gasarch and C. H. Smith. Recursion theoretic models of learning: some results and intuitions. *Annals of Mathematics and Artificial Intelligence*, 15:151–166, 1995.

[11] W. Gasarch and M. Velauthapillai. Asking questions versus verifiability. In *International Workshop on Analogical and Inductive Inference*, volume 642 of *Lecture Notes in Computer Science*, pages 197–213. Springer-Verlag, 1993.

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

[13] M. Kummer. The strength of noninclusions for teams of finite learners. In *Proc. 7th Annu. ACM Workshop on Comput. Learning Theory*, pages 268–277. ACM Press, New York, NY, 1994.

[14] Y. Matijasevic. Enumerable sets are diophantine (Russian). *Doklady Academy Nauk, SSSR*, 191:279–282, 1970. Translation in Soviet Math Doklady, Vol 11, 1970.

[15] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* McGraw Hill, New York, 1967.

[16] C. H. Smith. The power of pluralism for automatic program synthesis. *Journal of the ACM*, 29(4):1144–1165, October 1982. Was also in FOCS 1981.

[17] R. I. Soare. *Recursively Enumerable Sets and Degrees.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1987.