1980

The Power of Parallelism Automatic Program Synthesis

Carl H. Smith

Report Number:
80-350

Smith, Carl H., "The Power of Parallelism Automatic Program Synthesis" (1980). *Department of Computer Science Technical Reports*. Paper 281.
<https://docs.lib.purdue.edu/cstech/281>

The Power of Parallelism for Automatic Program Synthesis

Carl H. Smith

Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907

1. Introduction

The model of program synthesis studied below was essentially introduced in [8] and presented recursion theoretically in [3]. Herein we will consider program synthesis as performed by an *inductive inference machine* (abbreviated: IIM) which is an algorithmic device with no *a priori* bounds on how much time or memory resource it shall use, which takes as its input the graph of a function from N (the natural numbers) into N an ordered pair at a time (in any order), and which from time to time, as it is receiving its input, outputs computer programs. IIMs also serve as a model of the process by which an empirical scientist may algorithmically analyze a sequence of experimental results and conjecture an explanation of the phenomenon under investigation [8]. Due to the relevance to the philosophy of science, IIMs have, in one form or another, been studied for many years with and without mathematical rigor [1].

The results of [6] are extended to cover the case of program synthesis by finite collections of IIMs. Our motivation is three fold. First of all, we are interested in parallel computation. In the case of Turing machine computations, the introduction of parallelism does not enlarge the set of computable functions.

We view program synthesis as a potentially infinite process of learning by example. As such, IIMs are performing limiting computations. The results presented below indicate, in a very strong sense, that $n+1$ IIMs are capable of synthesizing programs for a strictly larger class of functions than can only n IIMs. We examine the tradeoffs between the number of IIMs used to synthesize a program and the proximity to complete correctness achieved by the synthesized program. Although more machines can always be employed to synthesize more accurate programs, increased error tolerance cannot be substituted for the lack of machines.

We are interested in developing a theory of algorithmic learning. By studying IIMs we are investigating the class of all algorithms which learn by example. The contribution towards this goal reported below is the precise description of the tradeoffs in learning power between the number of machines, the number of learning trials allowed, and the accuracy of the synthesized program. The definition of team learning (to be given precisely below) states that a program has been successfully synthesized by the team if there is an IIM in the team capable of successfully synthesizing the program. Despite our inability to determine which IIM in the team is successfully inferring a given function, there are situations where our notion of team learning is practical. Here is an example. Suppose we wish to send a collection of robots to investigate an alien planet. Each robot is equipped with identical data gathering and transmission equipment. The robots are distinguished by the particular learning program used as a subroutine to analyze the data collected to infer a program which is then used to predict events in the alien environment. It is plausible that a robot which successfully infers a reasonably accurate prediction program has a better chance of surviving when faced with a new situation which may be fatal to its existence (landslide, flood, etc.) than does a robot which is using a program

which almost always makes inaccurate predictions. We are concerned in particular with how many robots to send based on which criteria we use to determine successful inference of a prediction program. It will be shown that, in general, the larger the collection of suitably chosen robots we send, the larger the class of phenomena that can be correctly identified. Hence, it is reasonable to conclude that sending a larger suitably chosen collection of robots will increase the chances that at least one will survive to send back data.

A synthesized program is an explanation of the function input to the IIM in the sense that it can be used to predict the behavior of the function on arguments not yet seen by the IIM. An empirical scientist performs a series of experiments, records the corresponding results and conjectures an explanation of the phenomenon under investigation. The philosophers of science developed and studied various models of the scientific method. The first use of some of the recursion theoretic techniques that we use below applied to problems in inductive inference appears in [18]. However, in their study of the scientific method, the philosophers seem to regard a scientific community as a unified whole. Indeed, when several researchers are investigating the same phenomenon the results of every experiment are eventually known to all the researchers. Often there is no consensus on the proper explanation of the phenomenon under investigation. The theory presented below provides a model of the scientific method admitting several, possibly contradictory, opinions as to an explanation of some phenomenon. The applicability of our results to the philosophy of science may be slightly tenuous as we will regard an inference successful as soon as one scientist has found the correct explanation, regardless of the acceptance or rejection of the explanation by his or her colleagues.

2. Background and Notation

In this section we state several notational conventions, make some preliminary definitions, and review some results which are needed later on. Generally e and letters from the middle of the alphabet, ..., l, m, n range over N . Lower case letters from the beginning of the alphabet range over $(N \cup \{*\})$, where, by convention, $(\forall n \in N)[n < *]$. $\in, \subseteq, \subset$ denote respectively membership, containment, and proper containment for sets (including sets of ordered pairs). f ranges over total functions, ψ ranges over partial functions, and S ranges over subsets of R , the class of recursive functions. The sequence $\langle \varphi_i \rangle_{i \in N}$ denotes an arbitrary *acceptable numbering* of all and only the partial recursive functions [19, 20], also known as an *acceptable programming system* (see [12]). $\text{card}(A)$ denotes the cardinality of the set A . $\psi_1 =^n \psi_2$ (read: ψ_1 is an n -variant of ψ_2) means that $\text{card}(\{x | \psi_1(x) \neq \psi_2(x)\}) \leq n$, and $\psi_1 =^* \psi_2$ means that $\{x | \psi_1(x) \neq \psi_2(x)\}$ is finite. $\langle \dots \rangle$ denotes an arbitrary recursive function which maps $N \times N$ one-to-one onto N . $(\lambda x)[\psi(x)]$ denotes the function of x described by ψ . ϕ denotes the empty set with $\max(\phi) = 0$. Lower case Greek letters near the end of the alphabet, ..., σ, τ , ..., range over finite sequences of natural numbers, i.e., functions from finite initial segments of N into N . We sometimes speak of σ as an initial segment. $\lfloor m/n \rfloor$ denotes the greatest integer $\leq m/n$, and $\lceil m/n \rceil$ denotes the least integer $\geq m/n$.

There are several notions of what it means for an IIM to eventually succeed at synthesizing a program. The most popular notion, essentially due to [8], is given below.

DEFINITION 2.1.

An IIM M EX *identifies* a function f (written: $f \in EX(M)$) iff M , when fed the graph of f in any order, outputs but finitely many programs, the last one of which computes (or explains) f .

No restriction is made in Definition 2.1 that we should be able to algorithmically determine when (if ever) M on input from the graph of f has output its last computer program.

DEFINITION 2.2.

$M(\sigma)$ denotes M 's most recent output just after M has been fed the finite initial segment σ .

DEFINITION 2.3.

An IIM M is said to be *order independent* iff for any function f , if M 's last guess on some enumeration of the graph of f is p , then M 's last guess on every enumeration of the graph of f is p .

In the sequel we will only consider the synthesis of programs to compute total recursive functions. Clearly, any IIM M can be effectively transformed into an IIM M' which preprocesses the input graph of any recursive function f and feeds it to M in the order $(0, f(0)), (1, f(1)), \dots$. Hence, we will always assume without loss of generality, that all the IIMs we discuss are order independent. An order independence result which covers the case of partial recursive functions appears in [3]. Our results will hold, without modification, for the inference of partial recursive functions. However, the recursive functions are sufficient to illustrate all the distinctions we make below.

DEFINITION 2.4.

$M(f)$ denotes the last program M outputs when fed the graph of f if M outputs a last program; otherwise $M(f)$ is divergent.

If $M(f)$ is defined then we say M on f converges to program $M(f)$.

DEFINITION 2.5.

$EX = \{S \mid (\exists M)[S \subseteq EX(M)]\}$.

Definition 2.5 associates a class of sets of recursive functions with the inference criterion EX . We classify various inference criterion by set theoretically comparing the classes associated with each criteria. As the criteria become more general, the associated sets become larger. Allowing a finite number of errors in the program which an IIM converges to gives rise to several more identification criteria. Musa [14] shows that the mean time between software failures, for sufficiently complex systems, is an increasing but bounded function of the total of the run times of all the executions of the system. A reasonable conclusion to draw from Musa's work, which is consistent with observation, is that few large programs are totally bug free. A philosophical motivation for studying the inference of programs which exhibit anomalous behavior on some inputs stems from several examples of physical theories which, although generally correctly, yield inaccurate predictions in some instances. In the definitions of below a and b denote members of $(N \cup \{\infty\})$.

DEFINITION 2.6.

$M \in EX^a$ identifies f (written: $f \in EX^a(M)$) iff M , on input from the graph of f converges and $\varphi_{M(f)} =^a f$.

DEFINITION 2.7.

$$EX^a = \{S | (\exists M)[S \subseteq EX^a(M)]\}.$$

Note that $EX^0 = EX$. Also considered is the number of times an IIM changes its most current output. Of interest here is the number of trials a learning program must go through before it successfully completes the learning process. Another trial is completed every time an IIM outputs a program which is different from its previous conjecture. Counting mind changes is not the same as determining the complexity of the synthesis process. An IIM may output the same conjecture several times in a row, and may take a varying amount of time

to produce each conjecture. At best, counting mind changes may be a first approximation of the complexity of the inference process. In [16] counting the number of mind changes enroute to convergence to a program in a given r.e. sequence is studied. Some complexity of inference notions are considered in [17]. The complexity of the inference of finite automata is examined in [9].

DEFINITION 2.8.

$M \text{ } EX_b^a \text{ identifies } f$ (written: $f \in EX_b^a(M)$) iff $M \text{ } EX^a$ identifies f after no more than b mind changes (no restriction if $b = *$).

DEFINITION 2.9.

$EX_b^a\{S(\exists M)[S \subseteq EX_b^a(M)]\}$.

Note that $(\forall a)[EX^a = EX^a]$.

THEOREM 2.1[6].

$(\forall a, b, c, d \in (N \cup \{*\})) [EX_b^a \subseteq EX_d^c \text{ iff } a \leq c \text{ and } b \leq d]$.

Hence, one cannot, in general, trade mind changes for anomalies, or vice versa. The EX_b^a classes form a lattice under containment which is isomorphic to the partial product lattice on $\omega+1 \times \omega+1$.

Also considered are criteria where the IIM does not necessarily converge to a single program, but rather to a sequence of programs. As long as the IIM eventually outputs nothing but programs to compute the input function, then a prediction strategy which always uses the IIM's most current conjecture will be behaviorally correct.

DEFINITION 2.10.

An IIM $M \text{ } BC^a \text{ identifies } f$ (written: $f \in BC^a(M)$) iff M fed f outputs over time an infinite sequence of programs $p_0, p_1, p_2, \dots$ such that $(\forall n)[\varphi_{p_n} =^a f]$.

DEFINITION 2.11.

$$BC^0 = \{S \mid (\exists M)[S \subseteq BC^0(M)]\}.$$

BC^0 was originally defined by Barzdin [2] who was acting on an observation of Feldman [7].

THEOREM 2.2[6].

$$EX^* \subset BC^0.$$

THEOREM 2.3[6].

$$(\forall n \in \mathbb{N})[BC^n \subset BC^{n+1}].$$

L. Harrington (private communication) has devised an IIM which BC^* synthesizes a program for every recursive function (see [6]).

Suppose I is an identification criterion. A finite collection of IIMs *covers* a set of recursive functions with respect to I iff each function in the set is I -identified by at least one of the IIMs in the given collection. For $\forall M_1, M_2, \dots, M_n$ IIMs, let $I(M_1, M_2, \dots, M_n)$ denote $(I(M_1) \cup I(M_2) \cup \dots \cup I(M_n))$. Considering different size collections of IIMs gives rise to the following.

DEFINITION 2.12.

Suppose I is an inference criterion and $n \in \mathbb{N}$. Then $C(n, I) =_{df} \{S \mid (\exists M_1, M_2, \dots, M_n)[S \subseteq I(M_1, M_2, \dots, M_n)]\}$.

The remaining definitions in this section are necessitated by the proof techniques used below. We construct programs to compute functions which are easily identified by some IIMs, but not by others. By various recursion theorems we construct functions which output programs to compute themselves. Sometimes these programs will be "hidden" in the dips, or counterexamples to the monotonicity of the range of the functions constructed in the sequel. Hence, we make the following

DEFINITION 2.13.

$$D(f) = \{x | x > 0 \text{ and } f(x) < f(x-1)\}.$$

Note that f is monotone nondecreasing iff $D(f) = \emptyset$. In the examples of teams of IIMs that we construct, the interaction between the IIMs is rather trivial. Typically, the machines decide *a priori* which portion of the input they each will examine and then all the machines run identical algorithms on the data they examine. The portion of the input function examined by an IIM is called a *ply*.

DEFINITION 2.14.

Suppose ψ is a partial function and n is a positive integer. For $j < n$, the j^{th} n -ply of ψ is the partial function $(\lambda x)[\psi(n \cdot x + j)]$.

Clearly, any partial function is completely determined by its set of n -plies. We will pay attention to the monotonicity of the plies of the functions we construct.

DEFINITION 2.15.

Suppose ψ is a partial function defined on a (possibly infinite) initial segment of $\mathbb{N}$.

a) We say that ψ *monotonically extends* σ iff $[[\sigma = \emptyset \text{ and } \psi \text{ is monotone nondecreasing}] \text{ or } [\sigma \subset \psi \text{ and } \sigma \neq \psi \text{ and } (\forall x \in \text{domain}(\psi - \sigma)) [\psi(x) \geq \psi(x-1)]]]$.

b) Suppose $m < n$. We say ψ *monotonically extends* σ *along the* n -plies $\leq m$ (written: $\sigma \subset_m^n \psi$) iff $(\forall j \leq m)[\text{the } j^{\text{th}} \text{ } n\text{-ply of } \psi \text{ monotonically extends the } j^{\text{th}} \text{ } n\text{-ply of } \sigma]$.

Of course, when we diagonalize out of the set of functions identified by some collection of IIMs, we use an arbitrary collection to include all cases of teams of IIMs which are interacting in any way algorithmically possible. Sometimes, in performing the diagonalizations, it is necessary to describe the behavior of several programs which are working in parallel to baffle a given collection of

IIMs. These programs necessarily communicate with one another, but considering the details of how they communicate is avoided by the application of various recursion theorems.

3. Relationships between *EX* and *BC* Team Synthesis

In this section we establish several hierarchies of inference criteria based on the number of IIMs used in the inference process. The results in this section validate a "critical mass" principle for mechanized inductive inference. For each $n > 1$ there are sets of phenomena which can be identified by n robot scientists, but no fewer. That is, $n+1$ IIMs have the potential to synthesize programs for a larger class of functions than can be synthesised by only n IIMs. Recall the scenario of sending robots to investigate an alien planet. The following theorem suggests that we should send a large *diverse* collection of robots to increase the chances that at least one of them survives to send back data. Hence, for mechanistic learning devices, diversity is the key to success, just as in the biological world. The following is also a generalization of the Non-Union Theorem of [3] to BC^m -identification.

THEOREM 3.1.

$$(\forall n \in \mathbb{N}) [(C(n+1, EX) - \bigcup_{m \in \mathbb{N}} C(n, BC^m)) \neq \emptyset].$$

Proof. Select $n \in \mathbb{N}$. Let $S_n = \{f | (\exists j \leq n)(\forall g) [\text{if } g \text{ is the } j^{\text{th}} \text{ } n+1\text{-ply of } f \text{ then } [D(g) \text{ is finite and } \varphi_{g(\max(D(g)))} = f]]\}$. Let $M_0, M_2, \dots, M_n$ be IIMs. For $i \leq n$, M_i , on input f , examines g , the $i^{\text{th}} \text{ } n+1\text{-ply}$ of f and outputs all values $g(x)$ such that $x \in D(g)$. Clearly, $S_n \in EX(M_0, M_2, \dots, M_n)$. It remains to show that S_n is not included in $\bigcup_{m \in \mathbb{N}} C(n, BC^m)$. We present the proof of this for $n=2$ only. Suppose m is a natural number and M_1 and M_2 are IIMs. We exhibit below a function $f \in (S_2 - BC^m(M_1, M_2))$. By implicit use of the recursion theorem [10] we obtain the following program e_0 .

Begin Program e_0 . Place M_1 and M_2 in a "priority" queue [11] with M_1 initially at the front. On input x , successively execute the stages $s \geq 0$ below until (if ever) $\varphi_{e_0}(x)$ is defined. $\varphi_{e_0}^s$ denotes the finite initial segment of φ_{e_0} determined prior to stage s . Let $\varphi_{e_0}^0 = \{(0, e_0)\}$; hence, the 0th 3-ply of φ_{e_0} evaluated at zero is a program for φ_{e_0} .

Stage s . Let M be the IIM currently at the front of the queue. Search for distinct natural numbers $x_0, x_1, \dots, x_m$ and finite initial segments σ and τ such that $\varphi_{e_0}^s \subset_0^3 \sigma \subset_0^3 \tau$ and $(\forall i \leq m)[x_i \in \text{domain}(\tau - \sigma) \text{ and } \varphi_{M(\sigma)}(x_i) \text{ converges} \neq \tau(x_i)]$. If and when such $x_0, x_1, \dots, x_m, \sigma$ and τ are found, set $\varphi_{e_0}^{s+1} = \tau$ and move M to the rear of the queue. (Then program $M(\sigma)$ does not compute an m -variant of φ_{e_0} .)

End stage s .

End Program e_0 .

Case (1). φ_{e_0} is total. Let $f = \varphi_{e_0}$. The zeroth 3-ply of f is monotone nondecreasing and $f(3 \cdot 0 + 0) = e_0$ a program for f . Therefore, $f \in S_2$. Each member of the queue is at the front at infinitely many stages. Hence, for each IIM M in the queue, there are infinitely many distinct initial segments $\sigma \subset f$ such that $M(\sigma)$ does not compute an m -variant of f . Therefore, $f \notin BC^m(M_1, M_2)$.

Case (2). Not Case (1). Then φ_{e_0} is a finite initial segment. Let s be the first stage of program e_0 that never terminates. Then $\varphi_{e_0} = \varphi_{e_0}^s$. Suppose M is the IIM at the front of the queue upon entry into stage s of program e_0 . If σ is such that $\varphi_{e_0}^s \subset_0^3 \sigma$ then program $M(\sigma)$ converges on at most m arguments not in $\text{domain}(\sigma)$; otherwise, in stage s of program e_0 suitable corresponding $x_0, x_1, \dots, x_m$ and τ would be found and stage s would terminate. Hence M can-

not BC^m identify any function g such that $\varphi_{e_0}^s \subset_0^3 g$.

By another implicit application of the recursion theorem we obtain a program e_1 described as follows. Let x_0 be the least number not in $\text{domain}(\varphi_{e_0}^s)$. Choose the least number x_1 such that $3 \cdot x_1 + 1 > x_0$. Let $y_1 = 1 + \max(\text{range}(\varphi_{e_0}^s) \cup \{e_1\})$. Program e_1 is just like program e_0 except that M is deleted from e_1 's queue, $\varphi_{e_1}^0 = (\varphi_{e_0}^s \cup \{(x, y_1) \mid x_0 \leq x < 3(x_1 + 1) + 1\} \cup \{(3(x_1 + 1) + 1, e_1)\})$, and " $\subset_0^3$ " is replaced everywhere by " $\subset_1^3$ ". Therefore, $\varphi_{e_1}^s \subset_0^3 \varphi_{e_1}$.

Case (2.1). φ_{e_1} is total. Let $f = \varphi_{e_1}$. Let g be the first 3-ply of f . Then $\max(D(g)) = x_1 + 1$ and $g(x_1 + 1) = e_1$, a program for f . Therefore, $f \in S_2$. Then, since $\varphi_{e_0}^s \subset_0^3 f$, by the remark at the end of the first paragraph in the description of Case (2), $f \notin BC^m(M)$. By an argument similar to that of Case (1), $f \notin BC^m(M, M')$ for $M' \in (\{M_1, M_2\} - \{M\})$.

Case (2.2). Not case (2.1). Then φ_{e_1} is a finite initial segment. By an argument analogous to that in the first paragraph of the description of Case (2), for $M' \in (\{M_1, M_2\} - \{M\})$, M' does not BC^m -identify any recursive function g such that $\varphi_{e_1} \subset_1^3 g$. Furthermore, since any such g also monotonically extends $\varphi_{e_0}^s$ along the zeroth 3-ply we have that $g \notin BC^m(M_1, M_2)$. We complete this case by finding an $f \in S_2$ such that $\varphi_{e_1} \subset_1^3 f$.

By another implicit application of the recursion theorem we obtain a program e_2 described as follows. Let x_2 be the least number not in $\text{domain}(\varphi_{e_1})$. Choose the least number x_3 such that $3 \cdot x_3 + 2 \geq x_2$. Let $y_2 = 1 + \max(\text{range}(\varphi_{e_1}) \cup \{e_2\})$. φ_{e_2} agrees with φ_{e_1} on $\text{domain}(\varphi_{e_1})$, $(\forall x) [[\text{if } x_2 \leq x \text{ and } x \neq 3(x_3 + 1) + 2] \text{ then } \varphi_{e_2}(x) = y_2]$, and $\varphi_{e_2}(3(x_3 + 1) + 2) = e_2$. Let $f = \varphi_{e_2}$, a (total) recursive function. Clearly $\varphi_{e_1} \subset_1^3 f$; therefore, $f \notin BC^m(M_1, M_2)$. Let g be the second 3-ply of f . Then $\max(D(g)) = x_3 + 1$ and $g(x_3 + 1) = e_2$, a program for f .

Hence, $f \in S_2$.

This completes the proof that $S_2 \not\subset \bigcup_m C(2, BC^m)$. $S_n \not\subset \bigcup_m C(n, BC^m)$ can be proven similarly by applying the recursion theorem at most $n+1$ times to obtain programs $e_0, e_1, \dots, e_n$. If we do not fix m in advance before presenting programs $e_0, e_1, \dots, e_n$ and in stage s of programs $e_0, e_1, \dots, e_{n-1}$, we search for natural numbers $x_0, x_1, \dots, x_s$ instead of $x_0, x_1, \dots, x_m$. The function f we obtain works independently of m . ■

Two special cases of Theorem 3.1 have been previously discovered.

COROLLARY 3.2[2].

$$(C(2, EX) - C(1, BC)) \neq \phi.$$

COROLLARY 3.3[3].

$$(C(2, EX) - C(1, EX)) \neq \phi.$$

Theorem 3.1 establishes several hierarchies based on cover size.

COROLLARY 3.4.

$$(\forall a \in (N \cup \{*\})) [C(1, EX^a) \subset C(2, EX^a) \subset \dots \subset C(n, EX^a) \subset C(n+1, EX^a) \subset \dots].$$

COROLLARY 3.5.

$$(\forall m \in N) [C(1, BC^m) \subset C(2, BC^m) \subset \dots \subset C(n, BC^m) \subset C(n+1, BC^m) \subset \dots].$$

and [22] contain results about special cases where the union of two sets in EX is also in EX .

Allowing an finite unbounded number of anomalies also enables $n+1$ IIMs to synthesize in a single trial a class of functions which is not identifiable by n IIMs with respect to behaviorally correct inference.

THEOREM 3.6.

$$(\forall n \in \mathbb{N}) [C(n+1, EX_0^*) - \bigcup_{m \in \mathbb{N}} C(n, BC^m) \neq \emptyset].$$

Proof. Fix n . Let $S_n = \{f \mid (\exists x \leq n) [\varphi_f(x) = {}^*f]\}$. Clearly, $S_n \in C(n+1, EX_0^*)$. It remains to show that $S_n \not\subseteq \bigcup_{m \in \mathbb{N}} C(n, BC^m)$. Suppose $m \in \mathbb{N}$ and $M_1, M_2, \dots, M_n$ are IIMs. We exhibit below a function $f \in (S_n - BC^m(M_1, M_2, \dots, M_n))$. By implicit use of the $n+1$ -ary recursion theorem we obtain the following programs $e_0, e_1, \dots, e_n$.

Begin program e_i . If $i=n$ then $\varphi_{e_n} = \lambda x[0]$. Suppose that $i < n$. Place $M_1, M_2, \dots, M_n$ on a priority queue, in that order. On input x , successively execute the stages $s \geq 0$ below until (if ever) $\varphi_{e_i}(x)$ is defined. $\varphi_{e_i}^s$ denotes the finite initial segment of φ_{e_i} determined prior to stage s . Let $\varphi_{e_0}^0 = \{(i, e_i) \mid i \leq n\}$. For $0 < i < n$, set $\varphi_{e_i}^0 = \{(j, 0) \mid j \leq n\}$. σ_0^s denotes $\varphi_{e_0}^s$. For $0 < i < n$, $\sigma_{e_i}^s$ denotes $\sigma_{e_{i-1}}^s \cup \{(x, \varphi_{e_i}(x)) \mid x \in \text{domain}(\varphi_{e_i}^s - \sigma_{e_{i-1}}^s)\}$.

Stage s . Let M be the IIM currently in the $i+1^{\text{st}}$ position of the queue. Search for distinct natural numbers $x_0, x_1, \dots, x_m$ and finite initial segments τ and ρ such that $\sigma_i^s \subset \tau \subset \rho$ and $(\forall j \leq m) [x_j \in \text{domain}(\rho - \tau) \text{ and } \varphi_{M(\tau)}(x_j) \text{ converges } \neq \rho(x_j)]$.

Condition (1). A suitable $x_0, \dots, x_m, \tau$ and ρ are found before a similar $x_0, \dots, x_m, \tau$ and ρ are found in stage s of the construction of φ_{e_j} for any $j \neq i$. Set $\varphi_{e_i}^{s+1} = \varphi_{e_i}^s \cup \{(x, \rho(x)) \mid x \in \text{domain}(\rho - \varphi_{e_i}^s)\}$. Move M to the rear of the queue, without moving the first i IIMs in the queue.

Condition (2). An $x_0, \dots, x_m, \tau$ and ρ are found in stage s of the construction of φ_{e_j} for some $j \neq i$. If $j > i$ then $\varphi_{e_i}^{s+1} = \varphi_{e_i}^s$ and the search for $x_0, \dots, x_m, \tau$ and ρ in the construction of φ_{e_i} continues in stage $s+1$ at the same point where it was suspended in stage s . If $j < i$ then $\varphi_{e_i}^{s+1} = \varphi_{e_i}^s \cup$

$$\{(x, \varphi_{e_{i-1}}^{s+1}(x)) | x \in \text{domain}(\varphi_{e_{i-1}}^{s+1} - \varphi_{e_i}^s)\}$$

End stage s .

End program e_i .

Choose i least such that φ_{e_i} is total. Such an i exists since φ_{e_n} is total. If $i=0$, then let f be φ_{e_0} . Otherwise, choose the least stage s such that for $j < i$, $\varphi_{e_j} = \varphi_{e_j}^s$ and then set $f = \sigma_{i-1}^s \cup \{(x, \varphi_{e_i}(x)) | x \notin \text{domain}(\sigma_{i-1}^s)\}$. Since $f =^* \varphi_{e_i}$ and $f(i) = e_i$, $f \in S_n$. Suppose the queue at the beginning of stage s is $M_{k_1}, M_{k_2}, \dots, M_{k_m}$. We complete the proof by showing that $f \notin BC^m(M_{k_j})$, for each $0 < j \leq m$. Let M denote M_{k_j} .

Case (1). $j \leq i$. Then M is in position j of the queue at all stages $s' \geq s$. Hence, for all $\tau \supset \sigma_{j-1}^s$, $\varphi_{M(\tau)}$ is defined on at most m arguments $\notin \text{domain}(\tau)$, as otherwise, a suitable $x_0, x_1, \dots, x_m, \rho$, and τ would be found in some stage $s' > s$ in the construction of $\varphi_{e_{j-1}}$ and $\varphi_{e_{j-1}}$ would properly extend $\varphi_{e_{j-1}}^s$. Since $f \supset \sigma_{j-1}^s$, M on input f almost always outputs a program for a finite function. Therefore, $f \notin BC^m(M)$.

Case (2). $j > i$. Since φ_{e_i} is total, M is in the $i+1^{\text{st}}$ position of the queue at infinitely many stages. At each such stage past stage s , another τ is found such that $\tau \subset f$ and $\varphi_{M(\tau)}$ is not an m -variant of f . Therefore, $f \notin BC^m(M)$.

■

COROLLARY 3.7.

$$(\forall a \in (N \cup \{*\})) [C(1, EX_a^*) \subset C(2, EX_a^*) \subset \dots \subset C(n, EX_a^*) \subset C(n+1, EX_a^*) \subset \dots].$$

Corollary 3.5 is also a corollary of Theorem 3.6. Podnieks [15] used the sets S_n from the proof of Theorem 3.6 to separate BC type inference classes where the synthesis is successful if, in the limit, one out of every n programs output by the

IIM computes the input function. Podnieks established a hierarchy of such criteria parameterized by the size of the interval containing at least one correct program. The following Theorem indicates that even though $(C(2,EX)-BC) \neq \emptyset$, no matter how many IIMs are employed, no EX cover class can completely subsume the BC identifiable class of functions.

THEOREM 3.8.

$$(BC - \bigcup_{n \geq 0} C(n, EX^*)) \neq \emptyset.$$

Proof. Let S be $\{f \mid (\forall x)[\varphi_{f(x)} = f]\}$. Clearly $S \in BC$. We prove the $n=2$ case only. Let M_0 and M_1 be IIMs. By implicit use of the operator recursion theorem [4], we obtain a repetition free r.e. sequence of programs $e_0, e_1, p(0,0), p(0,1), p(1,0), \dots$ such that one of these programs computes an $f \in (S - EX^*(M_0, M_1))$. We proceed to give an informal effective construction of these functions in successive stages $s \geq 0$. e_1 is just a program for φ_{e_0} which differs from e_0 . Similarly, for each s , $p(1,s)$ is another program for $\varphi_{p(0,s)}$. $\varphi_{p(i)}^s$ denotes the finite initial segment of $\varphi_{p(i)}$ defined before stage s . $\varphi_{e_0}^0 = \emptyset$. $(\forall s) [\varphi_{p(0,s)}^0 = \varphi_{e_0}^s]$. e_0 initially places M_0 and M_1 on a queue.

Begin stage s . Let M be the IIM at the front of the queue. Set $q_s = M(\varphi_{e_0}^s)$.

Simultaneously execute the following three substages *until* (if ever) either suitable x and σ are found in substage (i) or a mind change is found in substage (ii).

(i) Dovetail a search for x and σ such that $\varphi_{p(0)}^s \subset \sigma$, $\text{range}(\sigma - \varphi_{e_0}^s) \subset \{e_0, e_1\}$, x is a member of $\text{domain}(\sigma - \varphi_{e_0}^s)$, and $\varphi_{q_s}(x)$ converges $\neq \sigma(x)$.

(ii) Let t be the step currently being executed in substage (iii). See if there is a τ such that $\varphi_{e_0}^s \subset \tau \subseteq$ what has been put into $\varphi_{p(t+2,s)}$ so far *and*

$M(\tau) \neq q_s$. ($\varphi_{p(t+2,s)}$ is made non empty by the inclusion of $\varphi_{e_0}^s$ in its graph in substage (iii) below.)

(iii) Execute more and more steps of the simultaneous construction of $\varphi_{p(0,s)}, \varphi_{p(1,s)} \dots \varphi_{p(i,s)}^t$ denotes the finite initial segment of $\varphi_{p(i,s)}$ determined prior to step t below. Set $\varphi_{p(0,s)}^0 = \varphi_{p(2,s)}^0 = \varphi_{e_0}^s$.

Begin step t . Let M' be the IIM next to the front of the queue. Set $\tau_t = M'(\varphi_{p(0,s)}^t)$. *Simultaneously* execute the following three substeps until (if ever) either suitable x and σ are found in substep (i) or a mind change is found in substep (ii).

(i) Dovetail a search for x and σ such that $\varphi_{p(0,s)}^t \subset \sigma$, $\text{range}(\sigma - \varphi_{p(0,s)}^t) \subset \{p(0,s), p(1,s)\}$, $x \in \text{domain}(\sigma - \varphi_{p(0,s)}^t)$, and $\varphi_{\tau_t}(x)$ converges $\neq \sigma(x)$.

(ii) See if there is a τ such that $\varphi_{p(0,s)}^t \subset \tau \subset$ what has been put into $\varphi_{p(t+2,s)}$ so far and $M(\tau) \neq \tau_t$. (Before step t , $\varphi_{p(t+2,s)}^t$ is made $= \varphi_{p(0,s)}^t$.)

(iii) Make $\varphi_{p(t+2,s)}$ have value $p(t+2,s)$ at more and more successive arguments not yet in its domain.

Condition (t.1). x and σ are found in substep (i) before a mind change is found in substep (ii). Set $\varphi_{p(0,s)}^{t+1} = \sigma$, do not extend $\varphi_{p(t+2,s)}$ any further, and set $\varphi_{p(t+3,s)}^{t+1} = \varphi_{p(0,s)}^{t+1}$.

Condition (t.2). A mind change is found in substep (ii) before or at the same time as suitable x and σ are found in substep (i). Set $\varphi_{p(0,s)}^{t+1} =$ what has been put into $\varphi_{p(t+2,s)}$ so far, make program $p(t+2,s)$ from this point on simulate program $p(0,s)$ on all inputs not yet in its domain so that $\varphi_{p(t+2,s)}$ will be the same as $\varphi_{p(0,s)}$, and set $\varphi_{p(t+3,s)}^{t+1} = \varphi_{p(0,s)}^{t+1}$.

End step t.

Condition (s.1). x and σ are found in substage (i) before a mind change is found in substage (ii). Set $\varphi_{p(0)}^{s+1} = \sigma$ and terminate step (iii). Move M to the rear of the queue.

Condition (s.2). A mind change is found in substage (ii) before or at the same time as suitable x and σ are found in substage (i). Set $\varphi_{e_0}^{s+1} =$ what has been put into $\varphi_{p(t+2,s)}$ so far, make programs $p(0,s)$ and $p(t+2,s)$ from this point on simulate program e_0 on all inputs not yet in its domain so that $\varphi_{p(t+2,s)} = \varphi_{p(0,s)} = \varphi_{e_0}$. Move M to the rear of the queue.

End stage s.

Case (1). Some stage s in the enumeration of φ_{e_0} never terminates. Suppose without loss of generality that e_0 's queue upon entry into stage s is M_0, M_1 .

Case (1.1). Some step t in the enumeration of $\varphi_{p(0,s)}$ never terminates. Then by substage (iii), $\varphi_{p(s+2)}$ is a (total) recursive function and $(\forall x) [\varphi_{p(s+2)}(x) = p(s+2)]$. Set $f = \varphi_{p(s+2)}$. Clearly $f \in S$. Program q_s is M_0 's last output on input f ; furthermore, q_s never converges on any x not in $\text{domain}(\varphi_{p(0)}^s)$ since, if it did, it could not converge to both e_0 and e_1 and so substage (i) would find suitable x and σ . It follows that φ_{q_s} is a finite function and hence not a finite variant of f . Similarly, τ_t, M_1 's last output on input from f , computes a finite function. Therefore, $f \in (S - EX^*(M))$.

Case (1.2) Not Case (1.1). Then $\varphi_{p(0,s)}$ is total. By Condition $t.1$, $p(0,s)$'s and $p(1,s)$'s are introduced into its range and these compute $\varphi_{p(0,s)}$; by Condition $t.2$, $p(t+2,s)$'s are introduced into its range but then $p(t+2,s)$ also computes $\varphi_{p(0,s)}$. Set $f = \varphi_{p(0,s)}$. Clearly, $f \in S$. Since stage s never terminates, $M_0(f)$ converges to $M_0(\varphi_{e_0}^s)$, a program which computes a finite function.

Suppose M_1 on f outputs a last program q . Then Condition $t.1$, holds at all but finitely many steps t . Hence, infinitely often, f is defined to differ from φ_q . Therefore $f \in S-(EX^*(M_0, M_1))$.

Case (2). Not Case (1). Then φ_{e_0} is a (total) recursive function and *everything* in its range is a program for φ_{e_0} ; By Condition $s.1$, e_0 's and e_0 's are introduced into its range and these compute φ_{e_0} ; by Condition $s.2$, $p(t+2,s)$'s are introduced into its range, but *then* $p(t+2,s)$ also computes φ_{e_0} . Set $f = \varphi_{e_0}$. Clearly, then, $f \in S$. Suppose M on f outputs a last program q . Then Condition $s.1$ holds at all but finitely many stages s . Hence, infinitely often, f is defined to differ from φ_q . Therefore, $f \in (S-EX^*(M_0, M_1))$. ■

The results of this section are sufficient to yield the following complete characterization of the relationships between the EX and BC cover classes.

THEOREM 3.9.

$(\forall m, n \in (N - \{0\})) (\forall a, b, c \in (N \cup \{*\})) [C(m, EX_b^a) \subset C(n, BC^c) \text{ iff } m \leq n \text{ or } c = * \text{ or } [a \neq * \text{ and } b \neq *]]$.

Proof. ($\Leftarrow$) If $m \leq n$ then $C(m, EX_b^a) \subset C(n, BC^c)$ by Theorem 2.2. If $c = *$, then $C(m, EX_b^a) \subset R \subset C(1, BC^c)$. Suppose $a \neq *$ and $b \neq *$. Choose m and $M_1, M_2, \dots, M_m$ arbitrarily. We will describe an IIM M such that $EX(M_1, M_2, \dots, M_m) \subset BC(M)$. We may suppose without loss of generality that each of $M_1, M_2, \dots, M_m$ output at most $b+1$ programs on any input. M on input σ simulates $M_1, M_2, \dots, M_m$, also on input σ . By simulating all programs output by $M_1, M_2, \dots, M_m$ on inputs in $domain(\sigma)$, for $card(\sigma)$ steps M can eliminate from consideration any program output by $M_1, M_2, \dots, M_m$ which converges incorrectly on more than a arguments. M then outputs a program e which mimics σ on all arguments in $domain(\sigma)$. Program e , on arguments not determined by σ , dovetails the computation of all programs

output by $M_1, M_2, \dots, M_m$ still under consideration, on the same argument, and mimics the output of the lexicographically least such program which converges first. Clearly, $EX(M_1, M_2, \dots, M_m) \subset BC(M)$.

($\Rightarrow$) We prove the contrapositive. Suppose $m > n$ and $c \neq *$. Then, if $a = *$, $(C(m, EX_b^a) - C(n, BC^c)) \neq \emptyset$ by Theorem 3.6. Finally, if $b = *$, $(C(m, EX_b^a) - C(n, BC^c)) \neq \emptyset$ by Theorem 3.1.

■

4. Tradeoffs between Mind Changes and Number of Machines

In this section we investigate the tradeoffs between the number of mind changes and the number of machines involved in the synthesis process. We will show that the number of mind changes can always be reduced by the employment of more IIMs. The following inclusion result, as will be seen, is best possible.

THEOREM 4.1.

$(\forall b, c \in \mathbb{N}) (\forall m, n \in (\mathbb{N} - \{0\})) (\forall a \in \mathbb{N} \cup \{*\})$ [if $m \leq n$ and $m \cdot (b+1) \leq n \cdot (c+1)$ then $C(m, EX_b^a) \subseteq C(n, EX_c^a)$].

Proof. Fix $b, c \in \mathbb{N}$, $m, n \in (\mathbb{N} - \{0\})$, and $a \in \mathbb{N} \cup \{*\}$. If $m = n$ or $b \leq c$ then the theorem follows immediately from Definition 2.8, so suppose $m < n$ and $c < b$. Let $M_1, M_2, \dots, M_m$ be IIMs. We may suppose without loss of generality that each of $M_1, M_2, \dots, M_m$ make at most b mind changes on any input. Then $M_1, M_2, \dots, M_m$ collectively output at most $m \cdot (b+1)$ different programs on any input. Let $p_1, p_2, \dots, p_t$ be their names in order of appearance as conjectures for some $t \leq m \cdot (b+1)$. If two conjectures are issued simultaneously by two different IIMs, then choose one arbitrarily to appear before the other. For the remainder of this proof, i will range over $\{1, \dots, m \cdot (b+1)\}$ and l will range over $\{1, \dots, m\}$. If

M_i outputs p_i and then outputs p_j , then $i < j$ and we say p_i is *irrelevant*.

We will now describe $M'_1, M'_2, \dots, M'_n$ such that $EX_b^a(M_1, M_2, \dots, M_m) \subseteq EX_c^a(M'_1, M'_2, \dots, M'_n)$. k will range over $\{1, \dots, n\}$. Each conjecture p_i will be output by exactly one of the M'_i 's. $num^i(M'_k)$ denotes $c+1$ minus *card* (the subset of $\{p_1, p_2, \dots, p_{i-1}\}$ output by M'_k), e.g. the number of additional guesses that M'_k can output and still EX_c^a identify the input function. $num^0(M'_k) = c+1$. Similarly, $num^i(M_l)$ denotes $b+1$ minus *card* (the subset of $\{p_1, p_2, \dots, p_{i-1}\}$ produced by M_l). $num^0(M_l) = b+1$. We say M'_k is *blocked* (at i) if either $num^i(M'_k) = 0$ or M'_k 's most recent conjecture is relevant. If M'_k is blocked and M'_k 's most recent conjecture was produced by M_l then we say M'_k is blocked by M_l . Suppose M_l outputs p_i . Look for an unblocked M'_k satisfying:

$$num^i(M'_k) = num^i(M_l) \quad (i)$$

If found, then M'_k not only outputs p_i but is also committed hereafter to output all and only the further (if any) outputs of M_l . We then say p_i is placed by exact match rule (i). If there are no unblocked M'_k 's satisfying (i) then let S be the set of unblocked M'_k 's satisfying:

$$num^i(M'_k) < num^i(M_l) \quad (ii)$$

Let $S' = \{M' | M' \in S \text{ and } (\forall M \in S)[num^i(M') \geq num^i(M)]\}$. If M_l was blocked at i by a member of S' , then that member of S' outputs p_i and we say p_i is placed by room to spare rule (ii). If M_l was not blocked at i by any member of S' , then choose the lexicographically least member of S' to output p_i . If $S' = \emptyset$ then look for an unblocked M'_k satisfying:

$$num^i(M'_k) > num^i(M_l) \quad (iii)$$

If there is an unblocked M'_k satisfying (iii) then choose the one with $num^i(M'_k)$ least, and that M'_k outputs p_i . We then say p_i is placed by not enough room rule (iii).

As each M'_k cannot output more than $c+1$ conjectures and will not output if its most recent conjecture is still relevant, it suffices to show that all conjectures are successfully placed by (i), (ii), or (iii). Since $c < b$, then p_i is placed by (ii).

Claim. If $p_1, p_2, \dots, p_{i-1}$ have all been placed by (i) or (ii) then p_i will also be placed.

Proof of Claim. Suppose the hypothesis of the claim. Suppose M_l conjectures p_i . If there is some M'_k which is not blocked at i , then either (i), (ii), or (iii) will hold for M'_k and M_l . Suppose by way of contradiction that p_i is not placed. Then $\forall M' \in \{M'_1, M'_2, \dots, M'_n\}$, M' is blocked at i . Choose $M' \in \{M'_1, M'_2, \dots, M'_n\}$ arbitrarily. We will show that either M' has already output $c+1$ conjectures or M' is blocked at i by some M such that if M' outputs all and only the future conjectures produced by M and M outputs $b+1$ conjectures, then M' would eventually output at least $c+1$ conjectures. If $\text{num}^i(M') = 0$, then M' has already output $c+1$ conjectures other than p_i . If $\text{num}^i(M') > 0$, then M' has already output $c+1 - \text{num}^i(M)$ conjectures and $(\exists l')[l' \neq l, M_{l'} \text{ is blocking } M']$. Since $p_1, p_2, \dots, p_{i-1}$ were placed by rules (i) or (ii) $M_{l'}$ may output at least $\text{num}^i(M')$ additional conjectures]. At any given point, each M_l can only block one M'_k . Hence $n \cdot (c+1)$ of $M_1, M_2, \dots, M_m$'s conjectures other than p_i have been accounted for. But, $1 + n \cdot (c+1) > m \cdot (b+1)$, a contradiction. This completes the proof of the claim. ■

We complete the proof of Theorem 4.1 by showing that (iii) can never be used to place any conjectures, in the case that $c < b$. Suppose by way of contradiction that (iii) is first used to place p_{i_0} which was produced by M_l . Suppose M'_k outputs p_{i_0} . Then, since $\text{num}^{i_0}(M'_k) > \text{num}^{i_0}(M_l)$, there is an $i_1 < i_0$ such that M_l produced p_{i_1} and

$$num^{i_0}(M'_k) = num^{i_1}(M_l). \quad (*)$$

One of two cases below holds.

Case (1). $num^{i_1}(M'_k) > num^{i_0}(M'_k)$. Then, $(\exists i_2 < i_1)$ [M_l produces p_{i_2} and $num^{i_1}(M'_k) = num^{i_2}(M_l)$], e.g. (*) holds with i_1 replaced by i_2 and i_0 replaced by i_1 .

Case (2). $num^{i_1}(M'_k) = num^{i_0}(M'_k)$. Then M'_k was blocked at i_1 , as otherwise M'_k would have output p_{i_1} , and hence p_{i_0} , by (i). Suppose $M_{l'}$ was blocking M'_k at i_1 . So, $M_{l'}$ must have output p_j with $i_1 < j < i_0$ removing the block before M'_k was selected to output p_{i_0} . Since $M_{l'}$ freed its block on M'_k , $(\exists k' \neq k)$ [$num^j(M'_{k'}) > num^j(M'_k) = num^{i_1}(M'_k)$]. Since $num^j(M'_{k'}) > num^{i_1}(M'_k)$, $(\exists i_2 < i_1)$ [M_l produced p_{i_2} and $num^j(M'_{k'}) = num^{i_2}(M_l)$], e.g. (*) holds with i_1 replaced by i_2 and i_0 replaced by j and k replaced by k' .

Starting from the assumption that M_l output p_{i_0} which was placed by (iii), we found an $i_1 < i_0$ such that M_l must have produced p_{i_1} and (*) held. By either Case (1) or Case (2) we found an $i_2 < i_1$ such that M_l must have produced p_{i_2} and (*) held. By repeating the above argument we can find $i_3 > \dots > i_{c+1}$ such that M_l produced $p_{i_0}, \dots, p_{i_{c+1}}$, a contradiction.

Therefore, each of $M_1, M_2, \dots, M_m$'s conjectures are placed by either (i) or (ii). Suppose $f \in EX_c^a(M_1, M_2, \dots, M_m)$. Then $(\exists l)$ [M_l 's last conjecture is an a -variant of f]. By the arguments above, the last conjecture output by M_l is also the last conjecture output by one of $M'_1, M'_2, \dots, M'_n$. Hence, $f \in EX_c^a(M'_1, M'_2, \dots, M'_n)$. □

Next we turn our attention to finding out how many machines need be traded to compensate for having fewer mind changes.

THEOREM 4.2.

$(\forall a, b \in \mathbb{N}) (\forall m, n \in (\mathbb{N} - \{0\}))$ [if $m \cdot (a+1) > n \cdot (b+1)$ then $C(m, EX_a^0) - C(n, EX_b^*) \neq \emptyset$].

Proof. Fix $a, b \in \mathbb{N}$ and $m, n \in (\mathbb{N} - \{0\})$. Let $S_l = \{f \mid 0 < \text{card}(D(f)) \leq l \text{ and } \varphi_f(\max(D(f))) = f\}$. Clearly, $S_{m \cdot (a+1)} \in C(m, EX_a^0)$. It remains to show that if $l > n \cdot (b+1)$ then $S_l \notin C(n, EX_b^*)$. Let $M_1, \dots, M_n$ be IIMs. We may suppose without loss of generality that each of $M_1, \dots, M_n$ output at most $b+1$ programs on any input. By implicit use of the $2^n \cdot l + 1$ -ary recursion theorem we obtain the following programs $e, p(0,0), \dots, p(0, 2^n - 1), p(1,0), \dots, p(l-1, 2^n - 1)$. Program e is constructed in at most l effective stages of finite extension below.

Begin program e . φ_e^s denotes the finite initial segment of φ_e determined prior to stage s . $\varphi_e^0 = \{(0, 1 + \max(e, p(0,0), \dots, p(0, 2^n - 1)))\}$. We let $i_0^s, \dots, i_{2^n - 1}^s$ denote canonical indices [20] of all the subsets of $\{M_j(\varphi_e^s) \mid 0 < j \leq n\}$. i_0^s will always denote the canonical index of the empty set. If there are fewer than k such subsets then i_k^s will also be a canonical index of $\emptyset$. x^s denotes the least number not in $\text{domain}(\varphi_e^s)$. For the remainder of the proof j will range over $\{1, \dots, n\}$.

Begin stage s . Simultaneously execute the following $2^n + 1$ substages (i), (ii.1), ..., (ii. $2^n - 1$), (iii) until (if ever) a mind change is found in substage (iii). In stage s , $k \in \{1, \dots, 2^n - 1\}$.

(i). Place $\varphi_e^s \cup \{x^s, p(s,0)\}$ into the graph of $\varphi_{p(s,0)}$, thereby adding a self-referential dip. Make $\varphi_{p(s,0)}$ have value $1 + p(s,0)$ on more and more successive arguments not yet in its domain.

(ii. k). $\varphi_{p(s,k)}^t$ denotes the finite initial segment of $\varphi_{p(s,k)}$ determined prior to step t . Set $\varphi_{p(s,k)}^0 = \varphi_e^s \cup \{(x^s, p(s,k))\}$, thereby placing a self-referential dip in the graph of $\varphi_{p(s,k)}$. Place the elements of the finite set with canonical index i_k^s on a queue in increasing order. If $i_k^s \neq \emptyset$ then successively execute the steps $t \geq 0$ below.

Step t. Let τ be the program currently at the front of the queue. Search for an x and a σ such that σ monotone nondecreasingly extends $\varphi_p^t(s,k)$, $x \in \text{domain}(\sigma - \varphi_p^t(s,k))$, and $\varphi_\tau(x)$ converges $\neq \sigma(x)$. If such x and σ are found set $\varphi_p^{t+1}(s,k) = \sigma$ and move τ to the rear of the queue.

End step t.

(iii). Search for j , k , and τ such that $\varphi_e^s \subset \tau \subset$ what has been put into $\varphi_p(s,k)$ so far and $[M_j(\varphi_e^s) \neq M_j(\tau) \text{ or } M_j \text{ outputs its first conjecture on } \tau]$.

If the execution of stage s is halted by the discovery of a j , k , and τ in substage (iii) above then let $\rho =$ what has been put into $\varphi_p(s,k)$ so far and set $\varphi_e^{s+1} = \rho \cup \{(\text{card}(\text{domain}(\rho)), 1 + \max \{p(s+1,0), \dots, p(s+1, 2^n-1)\})\}$.

End stage s.

End program e.

Let s be the last stage of program e that was attempted. Since stages end only when a new conjecture is found, $s < n \cdot (b+1)$. Then $\text{card}(D(\varphi_e^s)) = s+1$. Let $A = \{q \mid q = M_j(\varphi_e^s), \text{ and } \text{domain}(\varphi_q) \text{ is infinite}\}$. Choose k such that A is the finite set with canonical index i_k^s . Let $f = \varphi_p(s,k)$. f is a total function since each step in the construction of $\varphi_p(s,k)$ will terminate. Furthermore, $\max(D(f)) = p(s,k)$ and $\text{card}(D(f)) \leq s+2$. Hence, $f \in S_{1+n \cdot (b+1)}$. Since stage s never terminates, $M_j(f)$ converges to $M_j(\varphi_e^s)$. We conclude the proof by showing that $f \notin EX_b^*(M_j)$.

Case (1). $M_j(\varphi_e^s) \notin A$. Then $M_j(f)$ converges to a program which computes a finite function. Therefore, $f \notin EX_b^*(M_j)$.

Case (2). $M_j(\varphi_e^s) \in A$. Let $q = M_j(\varphi_e^s)$. Then q is at the front of $p(s,k)$'s queue at infinitely many steps. Hence, for infinitely many t , there is an $x \in (\varphi_p^{t+1}(s,k) - \varphi_p^t(s,k))$ such that $\varphi_q(x)$ converges $\neq f(x)$. Therefore, $f \notin EX_b^*(M_j)$.

COROLLARY 4.3.

$$(EX - \bigcup_{\substack{m>0 \\ a \in \mathbb{N}}} C(m, EX_a^*)) \neq \emptyset.$$

Proof. Let $S = \{f \mid D(f) \text{ is finite and } \varphi_{\max(D(f))} = f\}$. Clearly, $S \in EX$. Choose $m > 0$ and $a \in \mathbb{N}$. By Theorem 4.2, $S_{m \cdot (a+1)+1}$, a subset of S is not a member of $C(m, EX_a^*)$.

The results of this section, together with Theorem 3.6, are sufficient to completely characterize the tradeoff between mind changes and cover size in the cases where the synthesis of a finite variant of the input function is sufficient. Let $C(*, EX_a^*)$ denote $\bigcup_{n>0} C(n, EX_a^*)$, for any $a \in (\mathbb{N} \cup \{*\})$.

THEOREM 4.4.

$(\forall a, c \in (\mathbb{N} \cup \{*\}) - \{0\}) (\forall b, d \in (\mathbb{N} \cup \{*\})) [C(a, EX_b^*) \subseteq C(c, EX_d^*) \text{ iff } [(a \leq c \text{ and } b \leq d) \text{ or } [b \neq * \text{ and } [c = * \text{ or } d = *]] \text{ or } [a, b, c, d \in \mathbb{N} \text{ and } c \geq a \text{ and } c \cdot (d+1) \geq a \cdot (b+1)]]]$.

Proof ($\Leftarrow$). If $[a \leq c \text{ and } b \leq d]$ then clearly $C(a, EX_b^*) \subseteq C(c, EX_d^*)$. Suppose $[b \neq * \text{ and } c = *]$. Then by Theorem 4.1, $\forall m > 0, C(m, EX_b^*) \subseteq C(m \cdot (b+1), EX_d^*) \subseteq C(*, EX_d^*)$. Hence, $C(a, EX_b^*) \subseteq C(*, EX_d^*)$. Suppose $b \neq *, c \neq *$ and $d = *$. Then by Theorem 4.1, $(\forall m > 0), [C(m, EX_b^*) \subseteq C(c, EX_{m \cdot (b+1)}^*) \subseteq C(c, EX_*^*)]$. Hence, $C(a, EX_b^*) \subseteq C(c, EX_*^*)$. Now suppose that $a, b, c, d \in \mathbb{N}$, $c \geq a$, and $c \cdot (d+1) \geq a \cdot (b+1)$. Then, by Theorem 4.1, $C(a, EX_b^*) \subseteq C(c, EX_d^*)$.

($\Rightarrow$) We prove the contrapositive. If $a > c$, then by Theorem 3.6, $(C(a, EX_0^*) - C(c, EX_d^*)) \neq \emptyset$. Suppose, for the remainder of the proof, that $a \leq c$ and $b > d$. Then $d \in \mathbb{N}$. If $b = *$, then $S = \{f \mid D(f) \text{ is finite and } \varphi_{f(\max(D(f)))} = f\} \in EX_b^*$. By

Theorem 4.2, $\forall n > 0$, $\{f \mid 0 < \text{card}(D(f)) \leq n \cdot (d+1) + 1 \text{ and } \varphi_{f(\text{max}(D(f)))} =^* f\}$, a subset of S , is not an element of $C(n, EX_d^*)$. Hence, $S \in (C(a, EX_d^*) - C(c, EX_d^*)) \neq \phi$. If $c \in N$ then so is a . Suppose further that $c \cdot (d+1) < a \cdot (b+1)$. Then by Theorem 4.2, $(C(a, EX_b^*) - C(c, EX_d^*)) \neq \phi$.

□

5. Comparison of an unbounded number of Mind Changes with an unbounded number of Anomalies as traded off against the Number of Machines

In this section we compare team learning strategies with a finite unbounded number of trials to team learning strategies which are tolerant of a finite but unbounded number of errors. The symmetric relationship between mind change and anomaly bounds exhibited in Theorem 2.1 is also evident in the characterizations arrived at in this section.

THEOREM 5.1.

$(\forall a \in N) (\forall n \in (N - \{0\})) [EX_0^{n(a+1)} - C(n, EX^a) \neq \phi]$. *Proof.* Fix $a \in N$ and $n \in (N - \{0\})$. Let $S_{n(a+1)} = \{f \mid \varphi_{f(0)} =^{n(a+1)} f\}$. Clearly, $S_{n(a+1)} \in EX_0^{n(a+1)}$. Suppose $M_1, M_2, \dots, M_n$ are lms. It suffices to exhibit an $f \in (S - EX^a(M_1, M_2, \dots, M_n))$. Below we construct program e by implicit use of the recursion theorem. Program e will compute an $n(a+1)$ -variant of such an f . Program e employs n anomaly markers, each marking $a+1$ consecutive numbers we are temporarily trying to keep out of $\text{domain}(\varphi_a)$. We proceed to give an informal description of program e .

Begin Program e. On input x , successively execute the stages $s \geq 0$ described below until (if ever) $\varphi_s(x)$ is defined. φ_s^s denotes the finite amount of φ_s determined prior to stage s . Set $\varphi_s^0 = \{(0, e)\}$. Program e employs traveling anomaly markers $\alpha_1, \alpha_2, \dots, \alpha_n$ each marking $a+1$ consecutive integers. The integers marked by one α_j are disjoint from those marked by another. Let j and k range

over $\{1, 2, \dots, n\}$. A_j^s denotes the set of $a+1$ consecutive integers marked by α_j upon entry into stage s . Set $A_j^0 = \{1+(j-1)(a+1), 2+(j-1)(a+1), \dots, (a+1)+(j-1)(a+1)\}$. Clearly these sets are pairwise disjoint. At any point in the construction a number is *free* iff it is neither in $\text{domain}(\varphi_e)$ nor marked by any marker. τ^s , the *zero fill* of the finite function φ_e^s , is set $= (\varphi_e^s \cup \{ (x, 0) \mid (\exists j)[x \in A_j^s] \})$.

During the stages below, markers may be moved to least free numbers and the numbers they previously marked are placed in $\text{domain}(\varphi_e)$. Hence, $\text{domain}(\tau^s)$ is an initial segment. Let σ_j^s denote the largest initial segment of τ^s such that $(\text{domain}(\sigma_j^s) \cap A_j^s) = \emptyset$.

Let t range over $(\{-n, \dots, -2, -1\} \cup \mathbb{N})$. Initially α_j is assigned *task* $j-n-1$. For $t \geq 0$, *task* t may be assigned at stage t in the execution of program e to at most one marker α_j . For any t , *task* t assigned to α_j is the task of finding an $s \geq t$ such that either σ_j^s is consistent with φ_e and $(\exists x \in \text{domain}(\varphi_e^{s+1} - \varphi_e^s))$ [Program $M_j(\sigma_j^s)$ converges on $x \neq \varphi_e(x)$] or $(\exists \rho)[\sigma_j^s \subset \rho \subseteq \tau^s$ and ρ is consistent with φ_e and $M_j(\sigma_j^s) \neq M_j(\rho)$].

Each marker maintains a *pool* of tasks which are currently assigned to it. The *priority* of a task t is t . Priority t_1 is *higher* than priority t_2 iff $t_1 < t_2$. The *priority* of a marker α_j is the priority of the highest priority task in its pool at the beginning of stage s .

Marker α_j requires attention at stage s iff

- a) $(\exists x \in A_j^s)[\varphi_{M_j(\sigma_j^s)}(x)$ converges in $\leq s$ steps to 0] or
- b) $(\exists \rho)[\sigma_j^s \subset \rho \subseteq \tau^s$ and $M_j(\sigma_j^s) \neq M_j(\rho)]$.

At each stage at most one marker requiring attention will be selected *receive attention*. If α_j is selected to receive attention at stage s and α_j has priority t at s , then A_j^s will be placed in $\text{domain}(\varphi_e)$ and α_j will move during

stage s as part of an attempt to "complete" task t . Under such circumstances, t will be replaced by s in α_j 's pool.

Tasks can conflict with one another. Suppose markers α_j and α_k have respective priorities t_j and t_k at stage s where $\max(A_j^s) < \min(A_k^s)$, α_j has higher priority than α_k , and α_k receives attention. Then $A_j^s \subset \text{domain}(\sigma_k^s)$. At some later stage, α_j may require attention. The addition of A_j^s to $\text{domain}(\varphi_e)$ may cause σ_k^s to be inconsistent with φ_e in which case we say α_j injured α_k 's attempt to "complete" task t_j at stage s . Task t_k must then be reassigned to α_k . A system of *injury tags* is used to determine when a particular task must be reassigned to some marker. When a marker α_k with priority t receives attention at stage s and marker α_j has the potential to later injure α_k (i.e., α_j has higher priority than α_k and $A_j^s \subset \text{domain}(\sigma_k^s)$), then an injury tag (α_k, t) is placed on α_j . If at some later stage s , α_j does injure α_k , then the tag is removed and task t is reassigned to α_k .

We say a task t is *completed before stage s* iff either

- a) t is never assigned to any marker and $t < s$, or
- b) t is assigned to some α_j and $(\forall s' \geq s)[A_j^{s'} = A_j^s]$, or
- c) t is assigned to some α_j and t is not in α_j 's pool at or past stage s .

Stage s . Let α_j be the highest priority marker requiring attention. (If no marker requires attention, then set $\varphi_e^{s+1} = \varphi_e^s \cup \{(x, 0)\}$, where x is the least free number). α_j will now receive attention. Let t be α_j 's priority at this stage.

If a): $(\exists x \in A_j^s)[\varphi_{M_j(\sigma_j^s)}(x)$ converges = 0 in $\leq s$ steps], then do a.1) through a.5) below in order.

a.1) Set $\varphi_e^{s+1} = (\varphi_e^s \cup \{(x, 1) | x \in A_j^s\} \cup \{(x, \sigma_j^s(x)) | (\exists k) [x \in (A_k^s \cap \text{domain}(\sigma_j^s)) \text{ and } \alpha_j \text{ has higher priority than } \alpha_k]\})$.

a.2) If α_j has an injury tag (α_k, t') , reassign task t' to α_k .

a.3) In order of priorities, move each marker α_k such that $A_k^s \subset \text{domain}(\varphi_\sigma^{s+1})$ to the $\alpha+1$ least free numbers. Remove all injury tags from the markers just moved.

a.4) Place injury tags (α_j, t) on all markers α_k such that $A_k^s \subseteq \text{domain}(\sigma_j^s - \varphi_\sigma^{s+1})$. (Then α_k has higher priority than α_j at this stage.)

a.5) Remove t from α_j 's pool and assign task s to α_j .

If b): Not a) and $(\exists \rho)[\sigma_j^s \subset \rho \subset \tau^s \text{ and } M_j(\sigma_j^s) \neq M_j(\rho)]$, then do b.1) through b.4) below, in order.

b.1) Set $\varphi_\sigma^{s+1} = (\varphi_\sigma^s \cup \{(x, 0) \mid x \in A_j^s\} \cup \{(x, 0) \mid [x \in A_k^s \text{ and } (A_k^s \cap \text{domain}(\rho)) \neq \emptyset \text{ and } \alpha_j \text{ has higher priority than } \alpha_k]\})$.

b.2) In order of priorities, move each marker α_k such that $A_k^s \subset \text{domain}(\varphi_\sigma^{s+1})$ to the $\alpha+1$ least free numbers. Remove all injury tags from the markers just moved.

b.3) Place injury tags (α_j, t) on all markers α_k such that $A_k^s \subset \text{domain}(\rho - \varphi_\sigma^{s+1}) \neq \emptyset$. (Then α_k has higher priority than α_j at this stage.)

b.4) Remove t from α_j 's pool and assign task s to α_j .

End stage s.

End Program e.

We employ the following

LEMMA 5.2.

Each task is completed before some stage.

Proof of Lemma 5.2. We proceed by induction on task priorities. Task $-n$, the highest priority task, is assigned to α_1 . If α_1 never moves, then task n is completed before stage 0. Suppose α_1 moves for the first time during stage s . Since there are no higher priority tasks α_1 receives attention at stage s and no injury tags $(\alpha_1, -n)$ are placed at stage s . Therefore $-n$ cannot appear in α_1 's pool after stage s , and so task $-n$ is completed before stage $s+1$.

Suppose inductively that all tasks with priority higher than t are completed before state s_0 . If task t is never assigned to any marker or it is assigned to a marker which moves only finitely often, then task t will trivially be completed before some stage. Suppose that task t is assigned to a marker α_j which moves infinitely often. Choose the least stage $s_1 \geq s_0$ such that t is in α_j 's task pool at the beginning of stage s_1 . If no such s_1 exists, then task t is completed before stage s_0 . Pick the least stage $s_2 \geq s_1$ such that α_j moves during stage s_2 . Since all higher priority tasks are completed before stage s_0 , α_j requires and receives attention at stage s_2 . Hence, t is removed from α_j 's pool during stage s_2 . Suppose by way of contradiction that t is placed in α_j 's task pool during some stage past s_2 . Let s_3 be the least such stage. The reentry of t into α_j 's pool *must* be by step a.2) in stage s_3 since $s_3 > t$. Therefore, the marker α_k receiving attention at stage s_3 has an (α_j, t) injury tag. That injury tag was placed on α_k at some stage prior to s_2+1 when α_k had priority t' higher than priority t . α_k did not move during stages $s_2, \dots, s_3-1$, as otherwise the injury tag would have been removed prior to stage s_3 . Therefore, α_k had priority at least as high as t' upon entry into stage s_3 , and α_k moved during stage s_3 . Therefore, some task with priority at least as high as that of task t' was *not* completed before stage s_3 , contradicting the choice of s_0 .

■

We continue with the proof of Theorem 5.1. By steps a.1) and b.1) of the construction, $\text{domain}(\varphi_0)$ is cofinite. Clearly, if $x \notin \text{domain}(\varphi_0)$ then $(\exists j) (\forall s)[x \in A_j^s]$. Then $\text{card}(\text{domain}(N-\varphi_0)) \leq n(\alpha+1)$. Let A be the set of markers α_j such that $(\forall s)[A_j^s = A_j^{s+1}]$. Let t be the lowest priority task assigned to any marker in A . By Lemma 5.2 all tasks t' with priority at least as high as t are completed before some stage; hence, after this stage each marker in A has higher priority than any marker not in A . Choose stage s such that $(\forall \alpha_j \in A) (\forall \alpha_k \notin A) (\forall s' \geq s) [A_j^s = A_j^{s'} \text{ and } \alpha_j \text{ has higher priority than } \alpha_k \text{ at stage } s']$. Set $f = (\varphi_0 \cup \{ (x, 0) \mid \alpha_j \in A \text{ and } x \in A_j^s \})$. Clearly, f is a recursive function. Since $f(0) = e$, a program which computes an $n(\alpha+1)$ -variant of f , $f \in S_{n(\alpha+1)}$. We complete the theorem by proving that $f \notin EX^a(M_j)$

Case (1). $\alpha_j \in A$. Hence, but finitely many distinct tasks are assigned to α_j . Let t be the lowest priority task ever assigned to α_j . By Lemma 5.2 choose stage s so large that α_j does not move during or past stage s and all tasks with priority higher than t are completed before stage s . Hence, α_j does not require attention at or past stage s . Choose stage $s' \geq s$ such that for each marker α_k if $\max(A_k^s) < \min(A_j^s)$ and marker α_k moves during or after stage s , then α_k 's first move during or after stage s is before stage s' . Then, $\sigma_j^s \subset f$. If there were a ρ such that $\sigma_j^s \subset \rho \subset f$ and $M_j(\rho) \neq M_j(\sigma_j^s)$ then α_j would eventually require attention at or past stage s' . Therefore, $M_j(f)$ converges to $M_j(\sigma_j^s)$, a program which fails to compute f on any argument in A_j^s . Hence, $f \notin EX^a(M_j)$.

Case (2). $\alpha_j \notin A$. By Lemma 5.2 each of the infinitely many distinct tasks assigned to α_j is eventually completed. Suppose task t , assigned to α_j , is completed before a stage $s+1$ but no earlier. Then α_j has priority t and receives attention at stage s . Either there is an $x \in A_j^s$ such that $\varphi_{M_j(\sigma_j^s)}(x)$ converges to $0 \neq \varphi_0(x) = f(x)$ or there is a ρ such that $\sigma_j^s \subset \rho \subset \tau^s$ and $M_j(\rho) \neq M_j(\sigma_j^s)$. We claim that σ_j^s (or ρ if applicable) is contained in f . Any $x \in \text{domain}(\sigma_j^s - \varphi_0^{s+1})$ is

marked by some α_k which was given an (α_j, t) injury tag during step a.4) or b.4) of stage s . If α_j^s is not contained in f , then at some stage $s' > s$, $\varphi_e(x)$ was defined to be $1 \neq \sigma_j^s(x)$. Under these circumstances α_k would injure α_j restoring t to α_j 's pool by step a.2) of stage s' . This contradicts the choice of s . Hence, $\sigma_j^s \subset f$. Suppose $M_j(f)$ converges to q . Then infinitely many tasks assigned to α_j are completed by forcing $\varphi_e(x)$ to converge $\neq \varphi_q(x)$ which also converges, for some x temporarily marked by α_j . Hence $f \notin EX^*(M_j)$. This completes the proof of Theorem 5.1.

Remark. A simpler, more typical, priority scheme may be used in the case $n=2$. The following modification of the previous construction will suffice in the case where there are only two markers. Replace the use of tasks and injury tags by a marker queue. Priority is determined by proximity to the front of the queue. At the end of each stage the marker receiving attention (if any) is placed at the rear of the queue. All other aspects of the construction remain unchanged. Clearly, since only two markers are employed, the priority queue scheme ensures that, if both markers move infinitely often, then each marker makes infinitely many moves when it is at the front of the queue. However, when three markers are employed, one of them may move infinitely often without ever being at the front of the queue. Here is an example. Suppose α_1, α_2 and α_3 are on the priority queue in that order. If first α_2 receives attention, then α_1 , then α_2 again, then α_3 ; then the initial queue configuration is attained. If the previous sequence of moves is repeated infinitely often, then α_2 moves infinitely often without ever being at the front of the queue; hence, each of α_2 's moves may later be injured.

COROLLARY 5.3.

$$(EX_0^* - \bigcup_{\substack{m>0 \\ a \in N}} C(m, EX^a)) \neq \emptyset.$$

Proof. Let $S = \{f \mid \varphi_f(0) = {}^*f\}$. Clearly, $S \in EX_0^*$. By Theorem 5.1, $(\forall a \in N) (\forall n \in (N - \{0\})) [S_{n(a+1)}, \text{ a subset of } S, \text{ is not contained in } C(n, EX^a)]$.

■

Theorem 5.1 is generalized below in Theorem 5.4. The results of this and previous sections yield the following characterizations. Note the symmetry of the following two results.

THEOREM 5.4.

$$(\forall m, n \in (N - \{0\})) (\forall a, b \in (N \cup \{*\})) [C(m, EX^a) \subset C(n, EX_b^*) \text{ iff } m \leq n \text{ and } b = *].$$

Proof. ($\Leftarrow$) Immediate by definition 2.8.

($\Rightarrow$) We prove the contrapositive. If $m > n$, then by Theorem 3.1, $(C(m, EX^a) - C(n, EX_b^*)) \neq \emptyset$. If $b < *$, then by Corollary 4.3, $EX - C(n, EX_b^*) \neq \emptyset$.

■

THEOREM 5.5.

$$(\forall m, n \in (N - \{0\})) (\forall a, b \in (N \cup \{*\})) [C(m, EX_a^*) \subset C(n, EX^b) \text{ iff } m \leq n \text{ and } b = *].$$

Proof. ($\Leftarrow$) Immediate by Definition 2.8.

($\Rightarrow$) We prove the contrapositive. If $m > n$, then by Theorem 3.6, $(C(m, EX_a^*) - C(n, EX^b)) \neq \emptyset$. If $b < *$, then by Corollary 5.3, $EX_0^* - C(n, EX^b) \neq \emptyset$.

■

6. Tradeoffs Between Anomaly Bounds and Number of Machines

Our first theorem in this section shows that one can always trade anomalies for cover size. As will be shown, the following result is best possible.

THEOREM 6.1.

$$(\forall l \in \mathbb{N}) (\forall m, n \in (\mathbb{N} - \{0\})) [C(n, EX^l) \subseteq C(n \cdot m, EX^{\lfloor l/m \rfloor})].$$

Proof. Fix $l \in \mathbb{N}$ and $m, n \in (\mathbb{N} - \{0\})$. Suppose M is an IIM. It suffices to exhibit IIMs $M_0, M_1, \dots, M_{m-1}$ uniformly in M such that $EX^l(M) \subseteq EX^{\lfloor l/m \rfloor}(M_0, M_1, \dots, M_{m-1})$. For each $j < m$, M_j simulates M outputting a patched version of M 's most recent conjecture. M_j also dovetails the computation of M 's most recent conjecture on all arguments in the domain of the input seen so far. M_j then patches in correct answers for the least $j \cdot \lfloor l/m \rfloor$ numbers for which M 's most recent conjecture has not yet been discovered to converge correctly. Suppose $f \in EX^l(M)$. Let $p = M(f)$. Choose $k \leq l$ least such that $\varphi_p =^k f$. For $j < m$, $M_j(f)$ will converge iff $j \cdot \lfloor l/m \rfloor \leq k$.

Choose the largest $j < m$ such that $j \cdot \lfloor l/m \rfloor \leq k$. Then $M_j(f)$ converges to a program which computes f except at $(k - j \cdot \lfloor l/m \rfloor)$ anomalous inputs.

Case (1). $m \cdot \lfloor l/m \rfloor > k$. By the choice of j , $(k - j \cdot \lfloor l/m \rfloor) < \lfloor l/m \rfloor$. Therefore, $f \in EX^{\lfloor l/m \rfloor}(M_j)$.

Case (2). Not case (1). Therefore, $j = m - 1$ and $m \cdot \lfloor l/m \rfloor \leq k$. Since $k \leq l$, m divides l and $k = l$. Then $(k - j \cdot \lfloor l/m \rfloor) = \lfloor l/m \rfloor = \lfloor l/m \rfloor$. Hence, $f \in EX^{\lfloor l/m \rfloor}(M_j)$. ■

Recall that a stronger inclusion result holds (Theorem 4.1) when trading machines for mind changes instead of anomalies. We next turn our attention to determining when cover size cannot be traded for anomalies.

The *priority* of a task t is t . Priority t_1 is *higher* than priority t_2 iff $t_1 < t_2$. The *priority* of a marker α_j is the priority of the highest priority marker in its pool at the beginning of stage s .

Marker α_j requires attention at stage s iff

- a) $(\exists x \in A_j^s)[\varphi_{M_j(\sigma_j^s)}(x)$ converges in $\leq s$ steps to $v_j^s]$ or
- b) $(\exists \rho)[\sigma_j^s \subset \rho \subseteq \tau^s \text{ and } M_j(\sigma_j^s) \neq M_j(\rho)]$.

At each stage at most one marker requiring attention will be selected *receive attention*. If α_j is selected to receive attention at stage s and α_j has priority t at s , then A_j^s will be placed in *domain* (φ_s) and α_j will move during stage s as part of an attempt to "complete" task t . Under such circumstances, t will be replaced by s in α_j 's pool.

Tasks can conflict with one another. Suppose markers α_j and α_k have respective priorities t_j and t_k at stage s where $\max(A_j^s) < \min(A_k^s)$, α_j has higher priority than α_k , and α_k receives attention. Then $A_j^s \subset \text{domain}(\sigma_k^s)$. At some later stage, α_j may require attention. The addition of A_j^s to *domain* (φ_s) may cause σ_k^s to be inconsistent with φ_s in which case we say α_j *injured* α_k 's attempt to "complete" task t_j at stage s . Task t_k must then be reassigned to α_k . A system of *injury tags* is used to determine when a particular task must be reassigned to some marker. When a marker α_k with priority t receives attention at stage s and marker α_j has the potential to later injure α_k (i.e., α_j has higher priority than α_k and $A_j^s \subset \text{domain}(\sigma_k^s)$), then an injury tag (α_k, t) is placed on α_j . If at some later stage s , α_j does injure α_k , then the tag is removed and task t is reassigned to α_k .

We say a task t is *completed before stage* s iff either

- a) t is never assigned to any marker and $t < s$, or

so as to satisfy clauses (1) and (2) below, where $\psi = (\varphi_e - \{(x, \varphi_e(x)) \mid x \in A_j^s\})$.

$$D(\psi) = D(\psi \cup \{(x, v_j^s) \mid x \in A_j^s\}) = D(\psi \cup \{(x, v_j^{s+1}) \mid x \in A_j^s\}). \quad (1)$$

$$[A_j^s = A_j^t \Rightarrow v_j^s = v_j^t]. \quad (2)$$

v_j^s is defined as follows. Choose the least stage $t_j^s \leq s$ such that α_j marks A_j^s upon entry into stage t_j^s . Select the permutation $w_1, w_2, \dots, w_{(m \cdot n)-1}$ of $\{1, 2, \dots, (m \cdot n)-1\}$ such that $(\forall j, k)[j < k \Rightarrow \max(A_{w_j}^{t_j^s}) < \min(A_{w_k}^{t_k^s})]$. If $j = w_1$, then v_j^s is set = $\max(\{\varphi_e^{t_j^s}(x) \mid x < \min(A_j^{t_j^s})\})$. If $j = w_{k+1}$, then v_j^s is set = $\max(\{\varphi_e^{t_j^s}(x) \mid x \in \text{domain}(\varphi_e^{t_j^s}) \text{ and } x < \min(A_j^{t_j^s}) \cup \{1 + v_{w_k}^{t_k^s}\}\})$. Clearly each v_j^s satisfies clauses (1) and (2) above. τ^s , the *minimum fill* of the finite function φ_e^s , is set = $(\varphi_e^s \cup \{(x, v_j^s) \mid (\exists j)[x \in A_j^s]\})$. It turns out that

$$\text{domain}(\tau^s) \text{ is an initial segment} \quad (3)$$

At even numbered stages below, markers may be moved to least free numbers and the numbers they previously marked are placed in $\text{domain}(\varphi_e)$. At odd numbered stages, least free numbers are placed in $\text{domain}(\varphi_e)$. Hence, clause (3) is satisfied. Let σ_j^s denote the largest initial segment of τ^s such that $(\text{domain}(\sigma_j^s) \cap A_j^s) = \emptyset$.

Let t range over $(\{-(m \cdot n)+1, \dots, -2, -1\} \cup \mathbb{N})$. Initially α_j is assigned *task* $(-(m \cdot n)+j)$. For $t \geq 0$, *task* t may be assigned at stage t in the execution of program e to at most one marker α_j . For any t , *task* t assigned to α_j is the task of finding an $s \geq t$ such that *either* σ_j^s is consistent with φ_e and $(\exists x \in \text{domain}(\varphi_e^{s+1} - \varphi_e^s) [\text{Program } M_j(\sigma_j^s) \text{ converges on } x \neq \varphi_e(x)])$ *or* $(\exists \rho)[\sigma_j^s \subset \rho \subseteq \tau^s \text{ and } \rho \text{ is consistent with } \varphi_e \text{ and } M_j(\sigma_j^s) \neq M_j(\rho)]$.

Each marker maintains a *pool* of tasks which are currently assigned to it.

THEOREM 6.2.

$(\forall l, m, n \in \mathbb{N})$ [if $m \cdot n > 1$ then $(C(m, EX^{(n-1)l}) - C((m \cdot n) - 1, EX^{(l-1)}) \neq \emptyset]$.

Proof. Fix $l, m, n \in \mathbb{N}$ with $m \cdot n > 1$. Let $S = \{f \mid (\exists i < m)(\forall g)[g \text{ is the } i^{th} \text{ m-ply of } f \Rightarrow [D(g) \text{ is finite and } \varphi_{g(\max(D(g)))} = (n-1)^l f]]\}$. Clearly, $S \in C(m, EX^{(n-1)l})$. Suppose $M_1, M_2, \dots, M_{(m \cdot n)-1}$ are lIms. It suffices to exhibit an $f \in (S - EX^{(l-1)}(M_1, M_2, \dots, M_{(m \cdot n)-1}))$. Below we define an r.e. sequence of programs $e, p_1(0), p_1(1), \dots, p_2(0), p_2(1), \dots, p_{m-1}(0), p_{m-1}(1), \dots$ such that at least one of these programs computes a $(n-1)^l$ -variant of such an f . This sequence of programs is obtained by implicit use of the operator recursion theorem [4]. Program e employs the anomaly markers and priority queue scheme of Theorem 5.1. $(m \cdot n) - 1$ anomaly markers are employed, each marking l consecutive numbers we are temporarily trying to keep out of $\text{domain}(\varphi_e)$. The device of dips on plys from the proof of Theorem 3.1 is also used. The dips are, however, to values of the form $p_i(s)$. Program $p_i(s)$ computes a patched version of φ_e based on the first $2s + 1$ stages of program e . Programs $p_i(s)$ patch $l \cdot n \cdot i$ values. We proceed to give an informal description of program e .

Begin Program e. On input x , successively execute the stages $s \geq 0$ described below until (if ever) $\varphi_e(x)$ is defined. φ_e^s denotes the finite amount of φ_e determined prior to stage s . Set $\varphi_e^0 = (\{(0, e)\} \cup \{(i, p_i(0)) \mid 0 < i < m\})$, thereby initializing each m -ply of φ_e . Program e employs traveling anomaly markers $\alpha_1, \alpha_2, \dots, \alpha_{(m \cdot n)-1}$ each marking l consecutive integers. The integers marked by one α_j are disjoint from those marked by another. Let j and k range over $\{1, 2, \dots, (m \cdot n) - 1\}$. A_j^s denotes the set of l consecutive integers marked by α_j upon entry into stage s . Set $A_j^0 = \{m + (j-1)l, m + (j-1)l + 1, \dots, m + (j-1)l + (l-1)\}$. Clearly these sets are pairwise disjoint. At any point in the construction a number is *free* iff it is neither in $\text{domain}(\varphi_e)$ nor marked by any marker. With each set A_j^s there is an associated *minimum fill value* v_j^s . Each v_j^s will be defined

- b) t is assigned to some α_j and $(\forall s' \geq s)[A_j^{s'} = A_j^s]$, or
- c) t is assigned to some α_j and t is not in α_j 's pool at or past stage s .

Stage 2s. Stage 2s is the same as stage s from Theorem 5.1 with the following modifications. If no marker requires attention at stage 2s then set $\varphi_e^{2s+1} = \varphi_e^{2s}$. s is replaced everywhere by $2s$. 0 is replaced by σ_j^{2s} and 1 is replaced by σ_j^{2s+1} .

End stage 2s.

Stage 2s+1. As will be seen:

- (4) For $0 < i < m$ a dip to $p_i(s+1)$ is placed on the $n^{th}m$ -ply of φ_e iff at least one of the $n \cdot i$ highest priority markers at stage 2s moved during stage 2s.

Choose the permutation $w_1, w_2, \dots, w_{(m \cdot n)-1}$ of $\{1, 2, \dots, (m \cdot n)-1\}$ such that $(\forall j, k)[j < k \Rightarrow \alpha_{w_j}$ has higher priority than α_{w_k} at stage 2s]. Select the largest k such that $(\forall j \leq k)[A_{w_j}^{2s} = A_{w_j}^{2s+1}]$. Then the k highest priority markers at stages 2s did not move during stage 2s. Let $j_0 = \lfloor k/n \rfloor$. Let $v = 1 + \max(\text{range}(\tau^{2s+1}) \cup \{p_i(s+1) \mid 0 < i < m\})$. Next φ_e is defined on $x_0, x_1, \dots, x_{2m-1}$ the $2m$ least free numbers. We have, then

$$\varphi_e^{2s+1} \subset \varphi_e^{2s+2} \quad (5)$$

Set $\varphi_e^{2s+2} = (\varphi_e^{2s+1} \cup \{(x_u, v) \mid u < m \text{ or } (x_u \bmod m) = j_0\} \cup \{(x_u, p_i(s+1)) \mid m \leq u < 2m \text{ and } (x_u \bmod m) = i > j_0\})$. If the $n \cdot i$ highest priority markers at stage 2s did not move during stage 2s, then $k \geq n \cdot i$ and $j_0 \geq i$. Hence, no dips to $p_i(s+1)$ are placed on the i^{th} m-ply of φ_e . On the other hand, if one of $n \cdot i$ highest priority markers did move in the previous stage, the $k < n \cdot i$, $i > j_0$, and $(x, p_i(s+1))$ was placed in the graph of φ_e for some x on the i^{th} m-ply. Hence clause (4) is true.

End stage $2s+1$.

End Program e.

We employ the following

LEMMA 6.3.

Each task is completed before some stage.

Proof of Lemma 6.3. We proceed by induction on task priorities. Task $(-(m \cdot n)+1)$, the highest priority task, is assigned to α_1 . If α_1 never moves, then task $(-(m \cdot n)+1)$ is completed before stage 0. Suppose α_1 moves for the first time during stage s . Since there are no higher priority tasks α_1 receives attention at stage s and no injury tags $(\alpha_1, (-(m \cdot n)+1))$ are placed at stage s . Therefore $(-(m \cdot n)+1)$ cannot appear in α_1 's pool after stage s , and so task $(-(m \cdot n)+1)$ is completed before stage $s+1$.

Suppose inductively that all tasks with priority higher than t are completed before stage s_0 . If task t is never assigned to any marker or it is assigned to a marker which moves only finitely often, then task t will trivially be completed before some stage. Suppose that task t is assigned to a marker α_j which moves infinitely often. Choose the least stage $s_1 \geq s_0$ such that t is in α_j 's task pool at the beginning of stage s_1 . If no such s_1 exists, then task t is completed before stage s_0 . Pick the least stage $s_2 \geq s_1$ such that α_j moves during stage s_2 . Since all higher priority tasks are completed before stage s_0 , α_j requires and receives attention at stage s_2 . Hence, t is removed from α_j 's pool during stage s_2 . Suppose by way of contradiction that t is placed in α_j 's task pool during some stage past stage s_2 . Let s_3 be the least such stage. The reentry of t into α_j 's pool must be by step a.2) in stage s_3 since $s_3 > t$. Therefore, the marker α_k receiving attention at stage s_3 has an (α_j, t) injury tag. That injury tag was placed on α_k at

some stage prior to s_2+1 when α_k had priority t' higher than priority t . α_k did not move during stages $s_2, \dots, s_3-1$, as otherwise the injury tag would have been removed prior to stage s_3 . Therefore, α_k had priority at least as high as t' upon entry into stage s_3 , and α_k moved during stage s_3 . Therefore, some task with priority at least as high as that of task t' was *not* completed before stage s_3 , contradicting the choice of s_0 .

■

We continue with the proof of Theorem 6.2. By (3) and (5) and steps a.1) and b.1) of the construction, $\text{domain}(\varphi_e)$ is cofinite. We proceed to define the patched versions of φ_e . Suppose $s \in \mathbb{N}$ and $0 < i < m$. Let $\alpha_{j_1}, \alpha_{j_2}, \dots, \alpha_{j_{n \cdot i}}$ be the $n \cdot i$ highest priority markers at stage $2s-1$. Let $\psi = (\varphi_e - \{(x, \varphi_e(x)) \mid (\exists k)[0 < k \leq n \cdot i \text{ and } x \in (\text{domain}(\varphi_e) \cap A_{j_k}^{2s-1})]\})$. Set $\varphi_{p_i(s)} = (\psi \cup \{(x, v_{j_k}^{2s-1}) \mid (\exists k)[0 < k < n \cdot i \text{ and } x \in A_{j_k}^{2s-1}]\})$.

Clearly, if x is not in $\text{domain}(\varphi_e)$ then $(\exists j)(\bigvee s)[x \in A_j^s]$. Let A be the set of markers α_j such that $(\bigvee s)[A_j^s = A_j^{s+1}]$. Let t be the lowest priority task assigned to any marker in A . By Lemma 6.3 all tasks t' with priority at least as high as t are completed before some stage; hence, after this stage each marker in A has higher priority than any marker not in A . Choose stage s such that $(\forall \alpha_j \in A)(\forall \alpha_k \notin A)(\forall s' \geq s)[A_j^s = A_j^{s'} \text{ and } \alpha_j \text{ has higher priority than } \alpha_k \text{ at stage } s']$. Set $f = (\varphi_e \cup \{(x, v_j^s) \mid \alpha_j \in A \text{ and } x \in A_j^s\})$. Clearly, f is a recursive function. Note that by (1) all the dips along any m -ply of f are also dips along the same m -ply of φ_e . Select the largest $i < m$ such that $D(i^{\text{th}} m\text{-ply of } f)$ is finite. Such an i exists since the zeroth m -ply of f is monotone non-decreasing. Let g denote the $i^{\text{th}} m$ -ply of f . If $i = 0$, then by (4) there are $< n$ markers $\alpha_j \in A$. For $i = 0$, $D(g) = \emptyset$; hence, $g(\max(D(g))) = f(0) = e$. Therefore, since each α_j marks l integers, $\varphi_e = (n-1)^l f$. Hence, if $i = 0$, $f \in S$. Suppose $i > 0$. By (4) and the

choice of i , there is an s such that $g(\max(D(g)))=p_i(s)$. Again, by (4) and the choice of i , $i \cdot n \leq \text{card}(A) \leq (i+1) \cdot n$. By (2), $\varphi_{p_i(s)} \subseteq f$. Hence, the cardinality of $(N\text{-domain}(\varphi_{p_i(s)})) \leq (((i+1)n-1)-(i \cdot n))l = (n-1)l$. Therefore $\varphi_{p_i(s)} = {}^{(n-1)l}f$; hence, $f \in S$. We complete the theorem by proving that $f \notin EX^{l-1}(M_j)$.

Case(1). $\alpha_j \in A$. Hence, but finitely many distinct tasks are assigned to α_j . Let t be the lowest priority task ever assigned to α_j . By Lemma 6.3 choose stage s so large that α_j does not move during or past stage s and all tasks with priority higher than t are completed before stage s . Hence, α_j does not require attention at or past stage s . Choose stage $s' \geq s$ such that for each marker α_k if $\max(A_k^s) < \min(A_j^s)$ and marker α_k moves during or after stage s , then α_k 's first move during or after stage s is before stage s' . Then, by (2), $\sigma_j^s \subseteq f$. If there were a ρ such that $\sigma_j^s \subset \rho \subset f$ and $M_j(\rho) \neq M_j(\sigma_j^s)$ then α_j would eventually require attention at or past stage s' . Therefore, $M_j(f)$ converges to $M_j(\sigma_j^s)$, a program which fails to compute f on any argument in $A_j^{s'}$. Hence, $f \notin EX^{l-1}(M_j)$.

Case (2). $\alpha_j \notin A$. By Lemma 6.3 each of the infinitely many distinct tasks assigned to α_j is eventually completed. Suppose task t , assigned to α_j , is completed before a stage $s+1$ but no earlier. Then α_j has priority t and receives attention at stage s . Either $(\exists x \in A_j^s) [\varphi_{M_j(\sigma_j^s)}(x) \text{ converges to } v_j^s \neq \varphi_0(x) = f(x)]$ or $(\exists \rho) [\sigma_j^s \subset \rho \subset f \text{ and } M_j(\rho) \neq M_j(\sigma_j^s)]$. We claim that σ_j^s (or ρ if applicable) is contained in f . Any $x \in \text{domain}(\sigma_j^s - \varphi_0^{s+1})$ is marked by some α_k which was given an (α_j, t) injury tag during step a.4) or b.4) of stage s . If α_j^s is not contained in f , then at some stage $s' > s$, $\varphi_0(x)$ was defined to be $v_k^{s'} + 1 = v_k^s + 1 \neq v_k^s = \sigma_j^s(x)$. Under these circumstances α_k would injure α_j restoring t to α_j 's pool by step a.2) of stage s' . This contradicts the choice of s . Hence, $\sigma_j^s \subseteq f$. Suppose $M_j(f)$ converges to g . Then infinitely many tasks assigned to α_j are completed by forcing $\varphi_0(x)$ to converge $\neq \varphi_g(x)$ which also converges, for some x temporarily marked by α_j . Hence $f \notin EX^*(M_j)$. This completes the proof of Theorem 6.2.

□

Theorem 5.1 can be obtained as a corollary of Theorem 6.2. We conclude this section by combining the above results to obtain a complete characterization of the tradeoffs between cover size and anomaly bounds for EX^a - identification. For any $a \in (N \cup \{*\})$, let $C(*, EX^a)$ denote $\bigcup_{n \in N} C(n+1, EX^a)$.

THEOREM 6.4.

$(\forall a, c \in (N \cup \{*\} - \{0\})) (\forall b, d \in (N \cup \{*\})) [C(a, EX^b) \subseteq C(c, EX^d) \text{ iff } [[a \leq c \text{ and } b \leq d] \text{ or } [b \neq * \text{ and } c = *] \text{ or } [a, b, c, d \in N \text{ and } c \geq a \cdot (1 + \lfloor b/(d+1) \rfloor)]]]$.

Proof ($\Leftarrow$). If $[a \leq c \text{ and } b \leq d]$ then clearly $C(a, EX^b) \subseteq C(c, EX^d)$. Suppose $[b \neq * \text{ and } c = *]$. Choose $n > 0$ such that $\lfloor b/n \rfloor \leq d$. By Theorem 6.1, for any $m > 0$, $C(m, EX^b) \subseteq C(m \cdot n, EX^{b/n}) \subseteq C(*, EX^d)$. Hence, $C(a, EX^b) \subseteq C(*, EX^d)$. Now suppose that $[a, b, c, d \in N \text{ and } c \geq a \cdot (1 + \lfloor b/(d+1) \rfloor)]$. Choose $n > 0$ such that $\lfloor b/n \rfloor \leq d$. If $n > 1$, then $d < \lfloor b/(n-1) \rfloor$; therefore, in any case, $n \leq 1 + \lfloor b/(d+1) \rfloor$. Hence, $a \cdot n \leq c$. By Theorem 6.1, $C(a, EX^b) \subseteq C(a \cdot n, EX^{\lfloor b/n \rfloor}) \subseteq C(c, EX^d)$.

($\Rightarrow$) We prove the contrapositive. If $a > c$, then by Theorem 3.1, $(C(a, EX^0) - C(c, EX^d)) \neq \emptyset$. Suppose, for the remainder of the proof, that $a \leq c$ and $b > d$. Then $d \in N$. If $b = *$, then $S = \{f \mid D(f) \text{ is finite and } \varphi_{f(\max(D(f)))} = * f\} \in EX^b$. By the proof of Theorem 6.2, with $m = 1$ and n a positive integer, $\{f \mid D(f) \text{ is finite and } \varphi_{f(\max(D(f)))} = n^{(d+1)} f\}$, a subset of S , is not an element of $C(n, EX^d)$. Hence, $S \in (C(a, EX^*) - C(c, EX^d))$. If b and c are natural numbers, then so is a . Suppose further that $c < a \cdot (1 + \lfloor b/(d+1) \rfloor)$. Choose the largest n such that $b \geq (d+1)(n-1)$. Since $b > d$, $n > 1$. By Theorem 6.2, $(C(a, EX^{(d+1)(n-1)}) - C((a \cdot n) - 1, EX^d)) \neq \emptyset$. Since $b < (d+1)n$, $n \geq 1 + \lfloor b/(d+1) \rfloor$. Hence, $(a \cdot n) - 1 \geq a \cdot (1 + \lfloor b/(d+1) \rfloor) - 1 \geq c$. Therefore, $(C(a, EX^b) - C(c, EX^d)) \neq \emptyset$.

□

Theorem 6.4 implies, for example, that for $k, l, m, n \in \mathbb{N}$, $C(m+1, EX^k) \subseteq C(n+1, EX^l)$ iff $(n+1) \geq (m+1) \cdot (1 + \lfloor k/(l+1) \rfloor)$. This latter formula surprisingly appears in other contexts. PEX_m^k is defined [5] to be the class of set S of recursive functions such that some Popperian IIM, EX_m^k identifies S . In [5] it is shown that $PEX_m^k \subseteq PEX_n^l$ iff $(n+1) \geq (m+1) \cdot (1 + \lfloor k/(l+1) \rfloor)$.

7. Conclusions

Program synthesis was modeled as an inductive inference process performed by IIMs. The tradeoffs between the number of IIMs trying to synthesis a program and the generality of the criteria used to determine success were examined. A "critical mass" principle was discovered. Theorems 3.1 and 3.6 indicate that there are sets of functions which cannot be BC identified by any collection of n IIMs, yet can be inferred by $n+1$ IIMs with respect to EX inference. A characterization of the relative power of team BC type synthesis verses EX type synthesis was given in Theorem 3.9. From Theorem 6.1 we know that the employment of more IIMs will guarantee the synthesis of programs with fewer errors. A characterization of the tradeoffs between the number of IIMs involved in the synthesis process and the number of error to be tolerated is given by Theorem 6.4. Similarly, more IIMs can be used to synthesis a program in a fewer number of trials (Theorem 4.1). Theorem 4.4 gives the complete characterization of the tradeoffs between the number of trials and the number of IIMs. The tradeoffs between the number of anomalies and the number of trials is given by Theorems 2.1, 5.4, and 5.5. The question of characterizing the precise tradeoffs for number of IIM verses number of anomalies for BC type inference remains open.

8. Acknowledgements

Some of the results presented above originally appeared in the author's doctoral dissertation [21] done under the direction of Dr. J. Case, to whom we are indebted for the suggestion of problems and the robot scenario. Financial support came from NSF grants MCS-7903912 and MCS-7704388. D. Boles wrote a useful program to display the various tradeoff results. Computer time was supplied by the Computer Sciences Department of Purdue University. My colleagues M. O'Donnell and P. Young were always willing to discuss the ideas and proofs which appear above.

References

1. ANGLUIN, D. AND SMITH, C. H., *A discrete theory of automatic program synthesis*, In preparation 1981.
2. BARZDIN, J., "Two theorems on the limiting synthesis of functions," pp. 82-88 in *Theory of Algorithms and Programs*, ed. Barzdin, Latvian State University, Riga, U.S.S.R. (1974). 210
3. BLUM, L. AND BLUM, M., "Toward a mathematical theory of inductive inference," *Information and Control* 28 pp. 125-155 (1975).
4. CASE, J., "Periodicity in generations of automata," *Mathematical Systems Theory* 8 pp. 15-32 (1974).
5. CASE, J. AND NGOMANGUELLE, S., *Refinements of inductive inference by pop-perian machines*, to appear. 1981.
6. CASE, J. AND SMITH, C., *Comparison of identification criteria for mechanized inductive inference*, Haven Publishing Co. (1981). to appear.
7. FELDMAN, J., "Some decidability results on grammatical inference of best programs," *Math Systems Theory* 10 pp. 244-262 (1972).
8. GOLD, E.M., "Language identification in the limit," *Information and Control* 10 pp. 447-474 (1967).
9. GOLD, E.M., "Complexity of automation identification from given data," *Information and Control* 37 pp. 302-320 (1978).
10. KLEENE, S., "On notation for ordinal numbers," *Journal of Symbolic Logic* 3 pp. 150-155 (1938).
11. KNUTH, D., *The Art of Computer Programming: Fundamental Algorithms*, Addison-Wesley, Reading, Mass. (1968).
12. MACHTEY, M. AND YOUNG, P., *An Introduction to the General Theory of Algorithms*, North-Holland, New York, New York (1978).
13. MINICOZZI, E., "Some natural properties of strong-identification in inductive inference," *Theoretical Computer Science* 2 pp. 345-360 (1976).
14. MUSA, J. D., "A theory of software reliability and its application," *IEEE Transactions on Software Engineering* SE-1(3) pp. 312-327 (1975).
15. PODNIEKS, K. M., "Comparing various concepts of function prediction part I," pp. 68-81 in *Theory Of Algorithms and Programs*, ed. Barzdin, Latvian State University, Riga, U.S.S.R. (1974). 210
16. PODNIEKS, K. M., "Computational complexity of prediction strategies," pp. 89-102 in *Theory Of Algorithms and Programs*, ed. Barzdin, Latvian State University, Riga, U.S.S.R. (1977). Russian
17. PUDLAK, P. AND SPRINGSTEEL, F. N., "Complexity in mechanized hypothesis formation," *Theoretical Computer Science* 8 pp. 203-225 (1979).
18. PUTNAM, H., "Probability and confirmation," in *Mathematics, Matter and Method*, Cambridge University Press (1975). Originally appeared in 1963 as a Voice of America Lecture.
19. ROGERS, H. JR., "Gödel numberings of partial recursive functions," *Journal of Symbolic Logic* 23 pp. 331-341 (1958).
20. ROGERS, H. JR., *Theory of Recursive Functions and Effective Computability*, McGraw Hill, New York (1967).

21. SMITH, C.H., *Hierarchies of identification criteria for mechanized inductive inference*, SUNY/Buffalo, N.Y. (1979). Ph.D. Dissertation.
22. WIEHAGEN, R. AND LIEPE, W., "Characteristic properties of recognizable classes of recursive functions," *Elektronische Informationsverarbeitung und Kybernetik* 12 pp. 421-438 (1974). German