Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
All Rights Reserved by Academic Press, New York and London

Vol. 18, No. 3, June 1979
Printed in Belgium

Complexity Classes of Provable Recursive Functions*

DAN GORDON†

Departments of Mathematics and Computer Science, Purdue University, West Lafayette, Indiana 47907

Received July 6, 1977

Complexity measures and provable recursive functions (p-functions) are combined to define a p-measure as a measure for which Blum's axioms can be proved in a given axiomatic system. For p-measures, it is shown that the complexity class of a p-function contains only p-functions and that all p-functions form a single complexity class. Various other classes and a variation of a complexity measure, all suggested by the notion of provability, are also investigated. Classical results in complexity theory remain true when relativized to p-functions.

1. INTRODUCTION

The theory of provable recursive functions was studied by Patrick Fischer in [5]. Given a formal system S containing second order arithmetic, a recursive function is called provable (in S) if there exists an algorithm which computes it and which can be proved (in S) to be total. Such functions are usually called p-functions.

The class of p-functions is interesting because these are the functions we usually work with in practice. For example, part of the literature on numerical methods for (partial) differential equations consists of proofs that certain finite difference schemes converge, and this means that a computer program based on such methods is a p-function. Besides that, all well-written computer programs have built-in checks to make sure that they terminate, making them p-functions.

Another reason for studying p-functions is the not-unjustified claim that the class of all partial recursive functions is far too large to be of real *practical* interest. This claim has led to the consideration of various smaller classes of functions among which p-functions have a natural place. Furthermore, the notion of provability has been of interest in recent years with the study of correctness and equivalence of programs.

We consider the class of p-functions from the aspect of abstract computational complexity. Typical questions which arise are the following: Do classical results in computational complexity remain true when relativized to p-functions? What can be said about the complexity class of a p-function? What about classes of functions defined by provable conditions?

* This is a revised version of [4] which was written while the author was a graduate student at the Technion, Israel Institute of Technology. The author is indebted to Azaria Paz and Eliahu Shamir for their advice and to Michael Rabin for suggesting this research.

† Present address: Department of Mathematical Sciences, University of Cincinnati, Cincinnati, Ohio 45221.

0022-0000/79/030294-10$02.00/0

When trying to answer these questions it becomes clear that if a meaningful connection is to be made between "provability" and complexity, then Blum's axioms for the complexity measure should be provable in S. We call such a measure a p-measure. Requiring our measure to be provable can be justified by the fact that we are dealing only with p-functions and that in practice complexity measures are provable. Note however that from S, we can construct a measure for which Blum's axioms are not provable in S; and, given such a measure, we can add a suitable formalization of "the measure obeys Blum's axioms" (see Def. 2.2) as an axiom and obtain a new system in which the measure is provable. P-measures were also considered by P. Young independently in [8].

The main results for complexity classes of p-functions are Theorems 1 and 2 below.

THEOREM 1. *If the complexity measure is provable then the complexity class of a p-function contains only p-functions.*

This means that if the complexity of an algorithm is smaller (almost everywhere) than a p-function, then the function computed by that algorithm is a p-function. Note that this does not imply that the given algorithm is itself provably total but rather that the same function can be computed by some (possibly different) provably total algorithm. A case in point is an example by Hartmanis and Stearns [7] of a non-provably total algorithm with run time of n^2. This run time is a p-function and so by Theorem 1 the function computed by that algorithm is a p-function.

From Theorem 1 and the Union Theorem [6, p. 467] we get:

THEOREM 2. *All p-functions form a single complexity class.*

Theorems 1 and 2 are proved in section 3. In section 4 we explain why classical results in complexity theory can be relativized to p-functions and bring a few examples.

Section 5 deals with various classes of functions which can be defined by provable conditions. In section 6 we turn to a slight variation of the notion of a complexity measure, suggested by the fact that although some computations do not halt, it may be possible to prove this and stop the computation.

2. DEFINITIONS

Following Fischer [5], let S be a formal axiomatic system containing second order arithmetic and at least the following symbols:

connectors : $\wedge, \vee, \rightarrow, \sim$
quantifiers : $\exists, \forall$
predicate symbols : $\epsilon, =$
function symbols : $+, \cdot, s$ (successor function) and a function symbol for exponentiation.
individual constant : $\mathbf{0}$
individual variables : $x, y, z, \ldots$

Denote $s(\mathbf{0})$ by $\mathbf{1},\ldots,$ $s(\mathbf{n}-\mathbf{1})$ by $\mathbf{n}$ for every n.

Let $\phi_0, \phi_1, \phi_2, \ldots$ be an acceptable Gödel enumeration of all partial recursive functions. We denote the i'th machine by M_i. We say that i is an index for $f(n)$ if $\phi_i = f$.

As in [5], let $M(x, y, z, w)$ be a formula in S which expresses in S the primitive recursive relation of i, j, k, l iff the i'th algorithm applied to input j gives output k in not more than l steps.

DEFINITION 2.1. M_i is *provable* (in S) if

$$\vdash_S \forall y\, \exists z\, \exists w M(\mathbf{i}, y, z, w)$$

We write p-algorithm for provable algorithm. A function $f(n)$ is called provable (a p-function) if it can be computed by a p-algorithm.

Note that when we say "ϕ_i is a p-function" we do not mean that M_i is a p-algorithm, but rather that ϕ_i can be computed by some p-algorithm. By recursive function we always mean total recursive.

We further assume that S is sound for elementary number theory, i.e., no formula which is false in the standard model of arithmetic can be proved in S.

We recall that $\Phi_i(n)$ is a complexity measure [2] if:

(1) The function

$$\theta_\Phi(i, n, m) = \begin{cases} 1 & \text{if } \Phi_i(n) = m \\ 0 & \text{otherwise} \end{cases}$$

is recursive.

(2) Domain Φ_i = Domain ϕ_i.

DEFINITION 2.2. A complexity measure Φ is called a provable complexity measure (p-measure) if there exists an index n_0 such that

(1) M_{n_0} is a p-algorithm,

(2) for all i, n, m, $\phi_{n_0}(2^i \cdot 3^j \cdot 5^m) = \theta_\Phi(i, n, m)$,

(3) $\vdash_S \forall x\, \forall y[\exists z\, \exists w M(\mathbf{n}_0, \mathbf{2}^x \cdot \mathbf{3}^y \cdot \mathbf{5}^z, \mathbf{1}, w) \leftrightarrow \exists z\, \exists w M(x, y, z, w)]$.

(1) and (2) imply that θ_Φ is a p-function. (3) implies that "Domain Φ_i = Domain ϕ_i" is a theorem of S.

Unless stated otherwise, we shall assume from now on that *Φ is a fixed provable complexity measure*, and that n_0 is a fixed index for θ_Φ as above.

DEFINITION 2.3. (a) We say that a condition on n (natural number) is true almost everywhere (a.e.) if it is true for all but finitely many values of n.

(b) A condition on n is true infinitely often (i.o.) if it is true for infinitely many values of n.

DEFINITION 2.4. The complexity class of a function f is the set

$$C_f = \{\phi_i \mid \phi_i \text{ is total and } \Phi_i(n) \leqslant f(n) \text{ a.e.}\}$$

3. COMPLEXITY CLASSES OF P-FUNCTIONS

In this section we prove that the complexity of a p-algorithm is a p-function, and then prove Theorems 1 and 2.

THEOREM 3.1. *If M_i is a p-algorithm then $\Phi_i(n)$ is a p-function. Furthermore, a p-algorithm for $\Phi_i(n)$ can be effectively constructed from M_i.*

Proof. Let n_0 be an index for θ_Φ as in Definition 2.2. Define algorithm M which operates as follows for every input n: using M_{n_0}, M starts computing the sequence

$$\theta_\Phi(i, n, 0), \theta_\Phi(i, n, 1),..., \theta_\Phi(i, n, k),...$$

until the value 1 is obtained. If k is the first number such that $\theta_\Phi(i, n, k) = 1$, M gives k as output and stops. Obviously, M computes Φ_i. We shall prove that M is a p-algorithm:

(1) M_{n_0} is provable, therefore it is provable that for every m, the computation of $\phi_{n_0}(2^i \cdot 3^n \cdot 5^m)$ terminates, i.e. the computation of every element of the sequence terminates.

(2) M_i is a p-algorithm.

(3) "Domain ϕ_i = Domain Φ_i" is a theorem of S.

(2) and (3) imply (in S) that for every n there exists k such that $\theta_\Phi(i, n, k) = 1$. Therefore M is a p-algorithm.

The above is obviously an effective procedure for constructing M from M_i. ∎

THEOREM 1. *If f is a p-function and $g \in C_f$ then g is a p-function.*

Proof. Let M_j be a p-algorithm which computes f. Since $g \in C_f$, it follows that g is total and that there exists an index i for g such that $\Phi_i(n) \leqslant f(n)$ a.e. Therefore there exists a number N such that if $n > N$ then $\Phi_i(n) \leqslant f(n)$.

We shall now construct a provable algorithm M for g which operates as follows for any given input n:

(1) For $n \leqslant N$, M has the values of $g(n)$ in a table look-up, M prints $g(n)$ and halts.

(2) If $n > N$, M simulates M_j on n until it obtains the value $m = f(n)$.

(3) Using M_{n_0} (the p-algorithm for θ_Φ), M computes

$$\theta_\Phi(i, n, 0),..., \theta_\Phi(i, n, m).$$

(4) If in stage 3 all values for θ_Φ are different from 1, print 0 and halt. (This won't happen in our case, but 4 is necessary for M to be properly defined and provable.)

(5) If $\theta_\Phi(i, n, k) = 1$ for some $0 \leqslant k \leqslant m$, M simulates M_i on n. If and when M_i halts on n, M prints the result, $g(n)$, and halts.

For $n \leqslant N$, M obviously computes $\phi_i(n)$. For $n > N$, $\Phi_i(n) \leqslant f(n)$. Therefore for $k = \Phi_i(n)$, we get $0 \leqslant k \leqslant m$ and $\theta_\Phi(i, n, k) = 1$, and so the computation will proceed to stage 5. Therefore M computes $\phi_i = g$.

We prove that M is a p-algorithm by going through the stages of a computation:

(1) A table look-up is obviously a process which can be proved (in S) to terminate.

(2) M_j is a p-algorithm and so stage 2 can be proved in S to terminate.

(3) M_{n_0} is a p-algorithm and so the computation of a finite sequence, using M_{n_0}, can be proved to terminate.

(4) Immediate.

(5) Recall condition 3 of Definition 2.2 ("Domain Φ_i = Domain ϕ_i" is provable in S). Condition 3 implies (in S) that if $\theta_\Phi(i, n, k) = 1$ then M_i halts on n.

Therefore M is a p-algorithm. ∎

COROLLARY 3.2. *If ϕ_i is total and $\Phi_i(n) \leqslant \Phi_j(n)$ a.e., where M_j is a p-algorithm, then ϕ_i is a p-function.*

Proof. $\phi_i \in C_{\Phi_j}$ and by Theorem 3.1 $\Phi_j(n)$ is a p-function. Therefore by Theorem 1, ϕ_i is a p-function. ∎

Theorem 1 means that a function whose *complexity* is bounded by a p-function is also a p-function. In contrast, restricting the *value* of a function by a p-function does not make it a p-function, as shown by the following example:

EXAMPLE 3.3. The theorems of S can be recursively enumerated, and so the p-functions can be recursively enumerated. Let $p_0, p_1, \ldots$ be such an enumeration. Define

$$f(n) = \begin{cases} 0 & \text{if } p_n(n) = 1 \\ 1 & \text{o.w.} \end{cases}$$

$f(n)$ is recursive, $\{0, 1\}$-valued and different from every p-function.

For the proof of Theorem 2, we bring the Union Theorem as stated in [6].

UNION THEOREM. *Let $\{f_i \mid i = 0, 1, \ldots\}$ be a r.e. set of recursive functions such that for all i, n, $f_i(n) < f_{i+1}(n)$. Then there exists a recursive t such that $C_t = \bigcup_{i=0}^{\infty} C_{f_i}$.*

THEOREM 2. *There exists a recursive function t such that C_t is the set of all p-functions.*

Proof. As mentioned in Example 3.3, the theorems of S can be recursively enumerated.

Let $r(n)$ be a recursive enumeration of all p-algorithms, so $\{\phi_{r(i)} \mid i = 0, 1,...\}$ is the set of all p-functions.

We define a r.e. set of functions as follows:

$$f_0(n) = \Phi_{r(0)}(n)$$
$$f_{i+1}(n) = f_i(n) + \Phi_{r(i+1)}(n).$$

$\{f_i \mid i = 0, 1,...\}$ is a set of functions as required in the Union Theorem. By Theorem 3.1, $\Phi_{r(i)}$ is a p-function and so every f_i is a p-function. Therefore by Theorem 1, every complexity class C_{f_i} contains only p-functions.

On the other hand, if $g(n)$ is a p-function then for some i, $g = \phi_{r(i)}$. Now $\Phi_{r(i)}(n) \leqslant f_i(n)$ and therefore $g \in C_{f_i}$. Therefore $\bigcup_{i=0}^{\infty} C_{f_i}$ is the set of all p-functions, and the conclusion follows from the Union Theorem. ∎

4. Relativization of Complexity Theorems to P-Functions

In this section we look at some of the classical results in complexity theory and explain why they can be relativized to p-functions. By relativization of a theorem we mean the statement obtained from the theorem by replacing the words "complexity measure" and "total function" by the words "p-measure" and p-function" respectively.

The reason why relativized versions are usually true is that when the original theorems are of a "constructive" type, and the measure is provable and the given functions are provably total, then the proofs that the constructed functions are total can be given in S. Therefore the constructed functions are p-functions. Furthermore, if (some of) the assumptions of the original theorems are provable in S then (some of) the consequences and properties of the constructed functions are also provable in S.

There is nothing unexpected in these observations and they are only brought in answer to probable questions concerning relativization.

Only a few examples of such relativized theorems are brought, and the reader should then have no difficulty in checking other results. The following is the relativized version of the existence of arbitrarily complex (a.e.) functions:

Theorem 4.1. *Let f be any p-function. There exists a p-function g such that if i is any index for g (M_i not necessarily provable) then*

$$\Phi_i(n) > f(n) \text{ a.e.}$$

Any of the proofs in [2] or [6] can be used to construct g and to prove (in S) that g is total. The following is the relativized version of the speed-up theorem as stated in [6]. However, for an extensive treatment of speed-up for p-functions and for provably equivalent programs, see [8].

Theorem 4.2. (speed-up for p-functions). *If Φ is a p-measure such that for every i*

and n, $\Phi_i(n) \geqslant n$, $r(n)$ a recursive function bounded by a p-function, then there exists a p-function g with the following property: If $\phi_i = g$ (M_i not necessarily a p-algorithm) then there exists an index j such that $\phi_j = g$, M_j is a p-algorithm and $\Phi_i(n) \geqslant r(\Phi_j(n))$ a.e.

The proof, given in [4], follows the lines of [6] and consists of checking that all parts of the proof can be carried out in S. The theorem is first proved for the measure $L_i(n)$ of number of tape cells used by M_i on input n. Since Blum's axioms for $L_i(n)$ can be proved in second order arithmetic which is contained in S, $L_i(n)$ is a p-measure. The result is then extended to any p-measure by use of the relativized version of the fact that any two measures are recursively related [6, Theorem 4].

Note that if in Theorem 4.2 we replace the requirement "$\phi_i = g$" by "ϕ_i is provably equivalent to g", then the proof that $\phi_j = g$ can be carried out in S, and so ϕ_j is also provably equivalent to g. Therefore g has a speed-up among its provably equivalent algorithms as well as among all p-functions. The next theorem shows that the boundedness of r is (almost) necessary:

THEOREM 4.3. *If r is monotonic increasing and not bounded by any p-function, then no p-function has a speed-up by factor r.*

Proof. Assume to the contrary that g is a p-function which has a speed-up. Let M_i be any p-algorithm for g and j another index for g such that $\Phi_i(n) \geqslant r(\Phi_j(n))$ a.e. If we assume, as in Theorem 4.2, that $\Phi_j(n) \geqslant n$ we get $\Phi_i(n) \geqslant r(n)$ a.e. By Theorem 3.1, $\Phi_i(n)$ is a p-function and so r is bounded a.e. by a p-function, which contradicts our assumption. (The "a.e." can be dropped because a total function equal a.e. to a p-function is also a p-function.) ∎

Is speed-up by factor r always effective? As shown by Blum [3], speed-up by factor r is not effective when r is sufficiently large, and the bound given in [3] is certainly low enough to include p-functions.

The following is a relativization of the Gap Theorem as stated in [6]:

THEOREM 4.4. *If $r(n)$ is a p-function such that $r(n) \geqslant n$ then there exists a p-function t, monotonically increasing such that $C_t = C_{r \circ t}$. Furthermore, the monotonicity of t can also be proved in S.*

5. Classes of Functions Defined by Provable Conditions

In this section we consider various possible classes of functions, all suggested by the notion of provability.

DEFINITION 5.1. Let ϕ_i be any total function. We define conditions (a), (b) and (1)–(4) on j and i as follows:

(a) M_j is a p-algorithm. (1) $\Phi_j \leqslant \phi_i$ a.e.

(b) ϕ_j is a p-function. (2) $\phi_j \in C_{\phi_i}$.

(3) $\underset{S}{\vdash} (\Phi_j \leqslant \phi_i$ a.e.)

(4) $\underset{S}{\vdash} (\phi_j \in C_{\phi_i})$.

Denote by P_k^a, P_k^b, $k = 1,\ldots, 4$, the set of all functions ϕ_j for which conditions (a), (b) respectively and k, hold. Obviously these are dependent on ϕ_i.

When we write $\vdash(\Phi_j \leqslant \phi_i$ a.e.) we mean that a suitable formalization of the following statement is provable: "The complexity of machine M_j (as given by M_{n_0}, see Definition 2.2) is less than or equal to, a.e., the function computed by machine M_i". $\phi_j \in C_{\phi_i}$ means $\exists l(\phi_l = \phi_j \wedge \Phi_l \leqslant \phi_i$ a.e.), and by $\phi_l = \phi_j$ we mean that the functions computed by machines M_l and M_j are equal.

THEOREM 5.2. (1) $P_1^b = P_2^b = P_2^a$.

(2) $P_3^a \subseteq P_1^a \subseteq P_2^a$.

(3) $P_3^a \subseteq P_3^b \subseteq P_4^b \subseteq P_2^a$.

(4) $P_3^a \subseteq P_4^a \subseteq P_4^b \subseteq P_2^a$.

Proof. The following is obvious: for all $1 \leqslant k \leqslant 4$, $P_k^a \subseteq P_k^b$ and $P_1^x \subseteq P_2^x$, $P_3^x \subseteq P_1^x$, $P_3^x \subseteq P_4^x$, $P_4^x \subseteq P_2^x$ where $x = a$ or $x = b$. (1) We have $P_2^a \subseteq P_2^b \supseteq P_1^b$. Let $\phi_j \in P_2^b \Rightarrow \phi_j$ is a p-function and $\phi_j \in C_{\phi_i} \Rightarrow \exists l(\phi_j = \phi_l \wedge \Phi_l \leqslant \phi_i$ a.e.) $\Rightarrow \phi_l$ is a p-function and $\Phi_l \leqslant \phi_i$ a.e. Therefore $\phi_j = \phi_l \in P_1^b$. Therefore $P_1^b = P_2^b$.

Since ϕ_j is a p-function, there exists an index k such that M_k is a p-algorithm and $\phi_k = \phi_j$. Therefore $\phi_k \in P_2^a$, i.e., $\phi_j \in P_2^a$. Therefore $P_2^a = P_2^b$.

(2)–(4) follow from (1) and the above-mentioned inclusion properties. ∎

We turn now to the question of whether the inclusion relations are proper. The next theorem gives some partial answers. We write $P_k^a(S)$, $P_k^b(S)$ to express the dependence of these sets on S.

THEOREM 5.3. (1) *P_3^a and P_4^a are r.e.*

(2) *If ϕ_i is a p-function then $P_2^a = C_{\phi_i}$, and if, further, C_{ϕ_i} is r.e., there exists an extension S' of S, also sound for elementary number theory such that*

$$P_4^a(S') = P_4^b(S') = P_2^a(S') = P_2^a(S).$$

(3) *There exists a p-measure Φ and a p-function ϕ_i such that P_2^a is not r.e. (and therefore $P_4^a \subsetneq P_2^a$).*

(4) *There exists a p-function ϕ_i such that $P_3^a \subsetneq P_1^a$.*

Proof. (1) The theorems of S form an r.e. set and therefore P_3^a and P_4^a are r.e.

(2) If ϕ_i is a p-function then C_{ϕ_i} contains only p-functions, therefore $C_{\phi_i} = P_2^b = P_2^a$. Therefore, any extension S' of S will leave P_2^a unchanged.

Assume now that C_{ϕ_i} is r.e., and let l be an index s.t. $C_{\phi_i} = \{\phi_{\phi_l(k)} \mid k = 0, 1, 2,\ldots\}$ For every $k = 0, 1,\ldots$, define A_k to be the following sentence: "$M_{\phi_l(k)}$ is total and $\phi_{\phi_l(k)} \in C_{\phi_i}$". This sentence can be formalized in S. Since ϕ_l is a recursive function, $\{A_k \mid k = 0, 1,\ldots\}$ is a recursive axiom-scheme. Let S' be the theory obtained from S by adjoining this axiom-scheme. S' is also sound for elementary number theory because every A_k is true. We show now that $P_2{}^a \subseteq P_4{}^a(S')$. Let $\phi_j \in P_2{}^a = C_{\phi_i}$. Therefore there exists k such that $\phi_j = \phi_{\phi_l(k)}$. But $\phi_{\phi_l(k)} \in P_4{}^a(S')$ and so $\phi_j \in P_4{}^a(S')$. Therefore $P_2{}^a \subseteq P_4{}^a(S')$. Together with part 4 of Theorem 5.2, this gives the required result.

(3) The existence of Φ and recursive t s.t. C_t is not r.e. is shown in [6, Theorem 11]. t is in fact 0, and is therefore a p-function. The proof that Φ is a complexity measure can be given in S, and therefore Φ is a p-measure. According to (2), $C_t = P_2{}^a$, therefore $P_2{}^a$ is not r.e.

(4) In [7, p. 21] an example is given of an algorithm whose run time is n^2, but for which there is no proof that it runs in time $<2^n$. The construction is such that the algorithm halts in exactly n^2 steps or exactly 2^n steps, so it is a p-algorithm. So for $\phi_i(n) = n^2$, the function computed by that algorithm is in $P_1{}^a$ but not in $P_3{}^a$. ∎

6. A Modified Complexity Measure

We now turn to a slight modification of the notion of complexity, suggested by the idea that the theorems of S can be recursively enumerated. Let M be a machine which has as input two integers i, n. M searches through the theorems of S until it finds a theorem which states that machine M_i does not halt on input n. To calculate $\phi_i(n)$, set machine M_i to work on input n and machine M to work (in parallel) on the pair (i, n). The computation stops if (and only if) either M_i halts on n, or M has discovered a proof that M_i will not halt on input n. Define the complexity $\Phi_i(n)$ to be the number of steps (or any other measure) required by M_i to compute $\phi_i(n)$ or the number of steps required by M to find the proof that M_i will not halt on input n. If M_i does not halt on input n and no such proof can be found, then $\Phi_i(n)$ is undefined.

If $\Phi_i(n)$ is defined, we can distinguish between the two cases (1) $\phi_i(n)$ defined, and (2) $\phi_i(n)$ undefined and "$\phi_i(n)$ undefined" is a theorem of S. Such a measure Φ would therefore obey the following axioms:

(1) Domain $\phi_i \subseteq$ Domain Φ_i.

(2) The function

$$\theta_\Phi(i, n, m) = \begin{cases} 1 & \text{if } \Phi_i(n) = m \\ 0 & \text{otherwise.} \end{cases}$$

is recursive.

(3) There is a partial recursive function $\eta_\Phi(i, n)$ such that if $\Phi_i(n)$ is defined then $\eta_\Phi(i, n)$ is defined and

$$\eta_\Phi(i, n) = \begin{cases} 1 & \text{if } \phi_i(n) \text{ is defined} \\ 0 & \text{otherwise.} \end{cases}$$

These are exactly the axioms for what G. Ausiello [1] called a "weak complexity measure." Basic properties of this type of measure can be found in [1]. The above example probably lends further justification to the study of weak complexity measures.

For related work see J. Hartmanis, Relations between diagonalization, proof systems, and complexity gaps, *Theor. Comput. Sci.* 8 (1979), 239–253.

References

1. G. Ausiello, Abstract computational complexity and cycling computations, *J. Comput. System Sci* 5 (1971), 118–128.
2. M. Blum, A machine-independent theory of the complexity of recursive functions, *J. Assoc. Comput. Mach.* 14 (1967), 322–336.
3. M. Blum, On effective procedures for speeding-up algorithms, *J. Assoc. Comput. Mach.* 18 (1971), 290–305.
4. D. Gordon, On the computational complexity of provable recursive functions, Preprint series No. MT-164, Technion, Israel Institute of Technology, August 1973.
5. P. C. Fischer, Theory of provable recursive functions, *Trans. Amer. Math. Soc.* 117 (1965), 494–520.
6. J. Hartmanis and J. E. Hopcroft, An overview of the theory of computational complexity, *J. Assoc. Comput. Mach.* 18 (1971), 445–475.
7. J. Hartmanis and J. E. Hopcroft, Independence results in computer science, *ACM SIGACT News* 8 (Oct.–Dec. 1976), 13–24.
8. P. R. Young, Optimization among provably equivalent programs, *J. Assoc. Comput. Mach.* 24 (1977), 693–700.

Printed by the St. Catherine Press Ltd., Tempelhof 37, Bruges, Belgium