ZW

stichting
mathematisch
centrum

MC

AFDELING ZUIVERE WISKUNDE

ZN 60/75 MEI

P. VAN EMDE BOAS

TEN YEARS OF SPEEDUP

Prepublication

ZW

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

AMS(MOS) subject classification scheme (1970): 68A20, 02F99

ACM -Computing Reviews- category: 5.25

KEY WORDS & PHRASES: *Speedup theorem, Machine-independent complexity theory, Blum measure, Gödel speedup in logic, infinitely-often speedup, leveling.*

TEN YEARS OF SPEEDUP

by

Peter van Emde Boas

ABSTRACT

The paper presents a survey on the speedup phenomenon in the machine-independent theory of recursive functions, the techniques used to prove its existence, its non-effectiveness, its generalizations, and the relations between the speedup in recursion theory, and similar phenomena in logic.

1. INTRODUCTION

For almost half a century the mathematical world knows that it is fighting a war against the unknown where final victory is impossible; certainly there will be some great successes, and enough small advances to justify to the outside world that progress is being made, but the original goals of HILBERT's program: proving consistency of mathematics by mathematical methods, were abandoned, under pressure of GÖDEL's incompleteness results about 45 years ago. People have accepted that there are beliefs and truths which will remain unproven forever, among those the consistency of their own work being not the least important one.

From the lost battle mathematicians gained a new insight in the role of effectiveness in mathematics. The concept of computability, without which computer science is unconceivable, is based on this research on the foundations of mathematics.

When the development of the electronical computer resulted into a tremendous increase of computing capabilities, the mathematical world had to reconsider its ideas on computation. Several calculations which had been assumed to be unfeasible became possible, whereas others remained as impossible as before.

It became clear that the concept of computability was meaningless if it were not accompanied by a corresponding notion of complexity. Efficiency became a guiding principle in the design of algorithms, and computer scientists started to develop algorithms for automatically improving the efficiency of other programs.

Clearly the ideal program optimizer is a program which produces for each program an "equivalent" program which is "optimal" (i.e. as efficient as possible). However,

if equivalence is taken in its most general sense (the two computed functions are equal), this program optimizer is as realistic as the machine which mechanically proves the consistency of mathematics. This was proven about 10 years ago by BLUM [5], whose speedup theorem states that there are computable functions which have no optimal programs at all.

Again there was a big gain from the lost battle; BLUM delivered to theoretical computer science the axiomatic tools needed to discuss matters related to complexity without dealing with some concrete machine model; this way Abstract Complexity Theory was born. And like the mathematicians who did not stop proving theorems 45 years ago, the program optimizers still are improving upon their optimizing compilers, keeping in mind a more moderate but realistic goal. (That these goals have to be very moderate indeed can be concluded from results like in ALTON [2].).

In this paper I like to present a survey on what Abstract Complexity Theory has achieved, in particular on the subject of the speedup phenomenon, which originally was its prime motivation. In the subsequent sections I will discuss topics like:

1) the techniques used to prove the existence of speedup;
2) characterizations of complexity sequences, and their implications for concrete machine models;
3) the non-effectiveness of speedup;
4) variants of speedup;
5) the relation between speedup and GÖDEL's speedup result in logic;
6) the relevance of i.o. speedups for decision procedures for decidable theories.

I will not discuss the highly specialized topics of speedup of enumerations by changing order (as discussed by YOUNG [24]), or the importance of speedable functions for embedding of partial orders in the hierarchy of complexity classes which was studied by ALTON [1].

Almost all results discussed in this survey have been gathered from the literature. By bringing them together I hope to enable the individual researcher to evaluate for himself whether speedup is relevant to his actual work or not.

2. NOTATIONS

In the sequel the pair of sequences $((\varphi_i)_i,(\Phi_i)_i)$ denotes a Blum measure (i.e. $(\varphi_i)_i$ is an acceptable Gödelnumbering, φ_i and Φ_i have the same domain for each i, and the predicate $\Phi_i(x) = y$ is recursive), which may be a Turingmachine time or tape measure or a completely abstract complexity measure, depending on the context. The functions φ_i are called programs, and the functions Φ_i are called run-times. We say φ_i computes f, or i is an index for f whenever $\varphi_i = f$. If not stated otherwise all functions are partial recursive functions.

The domain of a function f is denoted by $\mathcal{D}f$, and we write $f(x) < \infty$ ($f(x) = \infty$) for $x \in \mathcal{D}f$ ($x \notin \mathcal{D}f$). The inequality $f \leq g$ means $\mathcal{D}f \supseteq \mathcal{D}g$ and $g(x) \geq f(x)$ for all $x \in \mathcal{D}g$. Strict inequality $f < g$ means that moreover $g(x) > f(x)$ for all $x \in \mathcal{D}g$. The inequality $k \leq \infty$ for finite k and $\infty \leq \infty$ are understood to be true but $\infty \leq k$ is false unless $k = \infty$. Functional composition of f and g is denoted $f \circ g$ i.e. $f \circ g(x) = f(g(x))$.

We use the "almost everywhere" quantifiers $\overset{\infty}{\forall}$ and $\overset{\infty}{\exists}$. So $\overset{\infty}{\forall}x[P(x)]$ means "P(x) holds for all but finitely many x" and $\overset{\infty}{\exists}x[P(x)]$ denotes "P(x) holds for infinitely many x". We use the special notation $f \propto g$ for $\overset{\infty}{\forall}x[f(x) \leq g(x)]$; consequently the negation: not$(f \propto g)$ is equivalent to $\overset{\infty}{\exists}[f(x) > g(x)]$.

If f(x) is some expression in x then $\lambda x[f(x)]$ describes the function defined by this expression. The expression $\mu z[P(z)]$ denotes "the least z such that P(x)". We use the special symbols $I = \lambda x[x]$ and $Z = \lambda x[0]$.

Using a fixed pairing function $\langle x,y \rangle$, increasing in both arguments, and its coordinate inverses π_1 and π_2, we can interpretate one-variable functions to be many-variable functions; an occasional super index like in $\varphi^2_j(x,y) = \varphi_j(\langle x,y \rangle)$ indicates the use of this interpretation.

A transformation of programs σ is a total recursive function operating on the indices of programs. Such transformations are defined, implicitly using the S-n-m axiom and/or the recursion theorem, by writing a formula like:

$$\varphi_{\sigma(i)}(x) \Leftarrow P(i,x)$$

where P denotes some expression in i and x. Note that the above expression defines σ.

Throughout this paper we assume that $\Phi_i(x) \geq x$ for all i and x, unless stated otherwise; by this assumption bounds like $R(x,\Phi_i(x))$ which occur frequently in abstract complexity theory, may be replaced by the more simpler estimate $R(\Phi_j(x)) = (R \circ \Phi_j)(x)$. An example of a measure where a run-time which grows less than linearly makes sense, is the tape measure of off-line Turing machines.

A program φ_i is called R-honest when $\Phi_i \propto R \circ \varphi_i$.

According to the traditional definition a program size function s is a total recursive function, such that the sets $\{\varphi_i \mid s(i) \leq k\}$ are finite and canonically enumerable for each k. In this paper, however, the size of a program will always be its index.

3. THE SPEEDUP PHENOMENON

Let R be some total function satisfying $R > I = \lambda x[x]$. A program φ_j for some function f is called R-optimal provided $\Phi_j \underline{\propto} R \circ \Phi_i$ for each program φ_i for f. This relation means that, modulo an "overhead factor" R, no program for f is better than φ_j.

The example of Turing machines, where programs can be sped up by a linear factor,

shows that, as far as optimality is concerned, such factors R must be taken into account. On the other hand, assuming that $\lim_{x\to\infty} R(x)/x = \infty$, it makes sense to say that a program that sorts x arbitrary numbers in time O(xlogx) is R-optimal.

The speedup theorem tells that, no matter how big R is chosen, there always exist functions which have no programs which are R-optimal; each program can be replaced by one which is even R-better almost everywhere; formally:

THEOREM. (BLUM [5], speedup): *Let* R > I*; then there exists a total function* f *satisfying:*

$$\forall i[\varphi_i = f \Rightarrow \exists j[\varphi_j = f \text{ and } R\circ\Phi_j \propto \Phi_i]].$$

The relation expressed in the formula above is described as: "f has R-speedup".

Although this result is easy to state, there exist no "easy" proofs for it up to this moment; even the easy proof given by YOUNG [25] seems to be more complex than necessary. All proofs published so far contain some involved use of the recursion theorem wich obscures the underlying idea, the machine-dependent proof given by HARTMANIS & HOPCROFT [13] being a notable exception. (It is easy to see that the result is measure-independent because of recursive relatedness).

All proofs use the concept of a complexity sequence (although this concept is not always formally introduced). Let $(p_i)_i$ be a sequence of functions satisfying $\mathcal{D}p_i = \mathcal{D}f$ for each i. The sequence is called a *complexity sequence for* f provided
1) for each j there exists an index k for f such that $\Phi_k \propto p_j$;
2) for each program φ_k for f there exists an index j such that $p_j \propto \Phi_k$.

The two conditions together show that the sequence $(p_i)_i$ is cofinal with the sequence of run-times of f in the partial order $\propto$. From the behaviour of a complexity sequence for f one can derive when f has speedup. For example if $R\circ p_{i+1} \propto p_i$ then f has R-speedup. If for some total effective operator Γ (cf. ROGERS [19] or SCHNORR [22]) the complexity sequence satisfies $\Gamma(p_{i+1}) \propto p_i$ then f has Γ-operator speedup (assuming that Γ is monotonic).

The general idea behind all proofs of speedup goes as follows. For some would-be complexity sequence $(p_i)_i$ define a function f such that 2) becomes valid; this is obtained by a diagonalization process enforcing: "if $\Phi_i(x) \leq p_i(x)$ too often then $f \neq \varphi_i$".

Next, by inspection of the diagonalization process, one discovers that the information gathered by use of the "expensive" functions in the complexity sequence, i.e. the functions p_i with i < u, is essentially finite; this information could have been encoded as well in the finite control of some new machine, which can compute f without executing the more expensive subcomputations.

Assuming that the functions p_i can be computed by a sequence of honest programs one estimates next the run-time of the more clever program for computing f in terms

of the values of p_i for $i \geq u$; a (dirty) computation and a call to the recursion theorem closes the argument. Only in the proof of HARTMANIS & HOPCROFT [13] this final part is clean due to the fact that Turingmachine tape is re-usable.

Gathering all good ideas from the literature I propose the following machine-independent "three-line" proof of the speedup theorem.

PROOF. Define the transformation σ by:

$$\varphi_{\sigma(i,u,v)}(x) \Leftarrow \underline{\text{if}}\ x < \pi_1 v$$
$$\underline{\text{then}}\ \text{tablelookup}(x, \pi_2 v)$$
$$\underline{\text{else}}\ 1 + \max\{\varphi_j(x) \mid u \leq j \leq x\ \underline{\text{and}}\ \Phi_j(x) \leq \Phi_i(x-j)$$
$$\underline{\text{and}}\ \forall_{y<x}[j \leq y \Rightarrow \Phi_j(y) > \Phi_i(y-j)]\}$$
$$\underline{\text{fi}}$$

In the above expression "tablelookup" denotes a total function which considers its second argument to be the encoding of a finite table of function values, to be evaluated at the value of the first argument, returning zero if the first argument lies outside the domain of the table.

Assume for the moment that φ_i is total. From the description of σ one derives directly:

1) If $f := \varphi_{\sigma(i,0,0)} = \varphi_j$ and $x \geq j$ then $\Phi_j(x) > \Phi_i(x-j)$.
2) $\forall u \exists v[\varphi_{\sigma(i,u,v)} = \varphi_{\sigma(i,0,0)}]$.

1) follows by a diagonal argument, whereas 2) holds since each program φ_j contributes at most once to the value of the else-part.

Next one derives by a combining lemma argument (cf. [13])

3) $\exists S > I[S \text{ total } \underline{\text{and}}\ \forall i,u,v[\Phi_{\sigma(i,u,v)} \propto \lambda x[S(\max\{\Phi_i(y) \mid 0 \leq y \leq x-u\})]]]$.

If we write $p_j = \lambda x[\Phi_i(x-j)]$ we conclude that the assumption that for all x $\Phi_i(x+1) \geq R(S(\Phi_i(x)))$ (i.e. Φ_i is fast increasing) is sufficient to prove $R \circ \Phi_{\sigma(i,u,v)} \propto p_{u-1}$. In this case 1), 2) and 3) together show that $(p_j)_j$ is the complexity sequence of the function f which has R-speedup.

In order to complete the proof it is, therefore, sufficient to provide a single fast increasing run-time Φ_i. This run-time is obtained by a simple application of the recursion theorem:

Let h be an arbitrary total function, and define the transformation τ by:

$$\varphi_{\tau(i)}(x) \Leftarrow \underline{\text{if}}\ x = 0\ \underline{\text{then}}\ h(0)$$
$$\underline{\text{elif}}\ \Phi_i(x) \leq R(S(\Phi_i(x-1)))\ \underline{\text{then}}\ \varphi_i(x) + 1$$
$$\underline{\text{else}}\ h(x)$$
$$\underline{\text{fi}}$$

Now if i is a fixed-point of τ ($\varphi_{\tau(i)} = \varphi_i$) then it is easy to see that φ_i com-

putes h and that Φ_i satisfies $\Phi_i(x+1) \geq R(S(\Phi_i(x)))$. This completes the proof. □

4. CHARACTERIZATION OF COMPLEXITY SEQUENCES

The proof of the speedup theorem which we have described consists of a simultaneous construction of a function f together with a complexity sequence $(p_i)_i$ for it. Still the two constructs are not treated equivalently, which can be recognized from the fact that the given argumentation shows that whenever Φ_i is a sufficiently fast increasing run-time then the sequence of shifts of this run-time will be the complexity sequence of the function $f_i = \varphi_{\sigma(i,0,0)}$.

It is a reasonable question what kind of sequences $(p_i)_i$ can be complexity sequences for recursive functions. Answers to this question can be found in MEYER & FISHER [17] and in SCHNORR [22].

In this section we let to a program φ^2_ℓ correspond the sequence of functions $(p_i)_i = (\lambda x[\varphi^2_\ell(i,x)])_i$. Note that if the program φ^2_ℓ is R-honest then the functions p_i are R'-honest for some suitable function R'. If we now replace in the definition of $\varphi_{\sigma(i,u,v)}$ the expression $\Phi_i(x-j)$ by $\varphi^2_\ell(j,x)$ and $\Phi_i(y-j)$ by $\varphi^2_\ell(j,y)$, it is easy to see that $\varphi_{\sigma(\ell,u,v)}$ describes a diagonalization function for the would-be complexity sequence $(p_i)_i$. Moreover, in order that the computation of $\varphi_{\sigma(i,u,v)}(x)$ converges (assuming x to be outside of the domain of the table encoded in v), the values of $\varphi^2_\ell(j,y)$ for $u \leq j \leq y \leq x$ have to be defined. A combining lemma argument yields the following result:

LEMMA. (MEYER & FISCHER [17, lemma 5]): *There exists a total function* H *such that the assumption*

$$\forall i \exists u \overset{\infty}{\forall} x[\varphi^2_\ell(i,x) \geq H(\max\{\Phi^2_\ell(j,y) \mid u \leq j \leq y \leq x\})]$$

is sufficient to show that the sequence $(p_i)_i = (\lambda x[\varphi^2_\ell(i,x)])_i$ *is the complexity sequence of some total function* f.

MEYER & FISCHER [17] claim that for the Turing tape measure the function H may be chosen as $\lambda x[2^{2^x}]$, i.e. two-fold exponentiation. As a consequence they have the following result for this concrete measure:

THEOREM. *Let* R *be total and* H*-honest. Then there exists a function* f *such that:*

1) $R \propto \Phi_i$ *for each index* i *for* f,
2) *there exists an index* j *for* f *such that* $\Phi_j \propto \lambda x[\underbrace{R(R(\ldots(R(x))\ldots)}_{x \text{ times}}]$,
3) f *has self-composition speedup: if* $\varphi_i = f$ *there exists an index* j *such that* $\varphi_j = f$ *and* $\Phi_j \circ \Phi_j \propto \Phi_i$.

SCHNORR [22] gives an analogous characterization of complexity sequences, but

also a kind of converse result. His characterization uses the concept of honesty:

THEOREM. *There exists a function* H *such that for each* R *the assumptions that* φ^2_ℓ *is* R*-honest and that the derived sequence* $(p_i)_i$ *satisfies* $H \circ R \circ p_{i+1} \propto p_i$ *are sufficient to make* $(p_i)_i$ *the complexity sequence of some total function* f*. Conversely, to this function* H *there corresponds a function* S *such that for each function* f *having* S*-speed up an index* ℓ *and a function* R *exist such that* φ^2_ℓ *is* R*-honest and such that the sequence* $(p_i)_i$ *derived from* φ^2_ℓ *is a complexity sequence for* f *satisfying the condition* $H \circ R \circ p_{i+1} \propto p_i$.

This assertion means that the technicalities involved in characterizing complexity sequences can be satisfied naturally for functions which have a sufficiently strong speedup.

The following characterization of complexity sequences at the bottom of the Turing time measure is due to SCHNORR & STUMPF [21]:

THEOREM. *Let* φ^2_ℓ *be a program such that for some constant* c *and for all* n *and* x *the following assertions hold:*

1) $\log(x+1) \leq \varphi^2_\ell(n,x)$,
2) $\Phi^2_\ell(n,x) \leq c\ \varphi^2_\ell(n,x)$,
3) $\varphi^2_\ell(n,x) < \infty \iff \varphi^2_\ell(0,x) < \infty$,
4) $\lim_{y\to\infty}(\varphi^2_\ell(n,y)/\varphi^2_\ell(n+1,y)) = \infty$,

then the derived sequence $(p_i)_i = (\lambda x[\varphi^2_\ell(i,x)])_i$ *is the complexity sequence of some partial* 0 - 1 *valued function in the Turing time measure.*

This result shows that even programs having a run-time only slightly larger than $\log x$ can sometimes be sped up by more than a linear factor. The proof consists of a modification of the traditional speedup construction and a carefull analysis of its time requirements.

5. THE NON-EFFECTIVENESS OF SPEEDUP

Once having discovered the existence of faster programs it would be nice if one could effectively generate for each given program a new one which is faster; for many mathematicians existence without a construction method is as meaningfull as non-existence.

Let us consider the function f having a speedup as was constructed in section 3, together with its corresponding complexity sequence $(p_i)_i$. Clearly, if we want to find a program for f with a run-time bounded by p_u then any program $\varphi_{\sigma(i,u+1,v)}$ which computes the function (i.e., v must encode one of the infinitely many tables containing "sufficiently many" initial values of f) will be fine.

It is my experience that an uninitiated reader of a speedup proof gets the in-

correct impression that speedup is effective, by overlooking that such a "good" value of the parameter v is never constructed but only claimed to exist. The problem is that the table encoded in v must store the function values of f over an interval which is sufficiently long in order for those indices $j < u$ which ever contribute to the value of the else part ("by being cancelled" as this process is usually called), to do so at arguments within this interval. Since some indices will never be cancelled and since we cannot indicate which indices will behave so in advance, the size of the table (and consequently the right value of the parameter v) is unpredictable.

One may ask whether one needs to store the complete table of function values in the parameter v. Since expensive calculations are allowed on an initial segment it is sufficient to store only the length of the table, but this number is unpredictable as well. One can also store in v the total number of indices $< u$ which will be cancelled at any time; by updating the collection of cancelled indices the program discovers automatically when its expensive subcomputations of the functions $p_i(x)$ with $i < u$ may be omitted. Since this number v is bounded by u, this leads to the strange situation that the faster program is known to be one among a finite set of programs being about equally efficient, which compute functions having at most u incorrect values.

The above explanation is no formal proof of the assertion that speedup is not effective. Results on this subject have been published by BLUM [5,6], MEYER & FISCHER [17], HELM & YOUNG [14] and SCHNORR [20].

The oldest result on non-effectiveness of speedup is due to BLUM [5, corollary 1]. The result states that for sufficiently large R there exist no recursive enumeration of programs for a function f with R-speedup such that for each program φ_i for f some R-faster program φ_j occurs somewhere else in the enumeration (remember that the collection of all programs for a given function is never recursively enumerable). Erroneously BLUM states that this assertion implies that speedup is not effective, which mistake he confesses in [6].

The proof is based on the argument, that such a sequence of programs may be run in parallel yielding a program which is faster than all programs in the sequence.

In [6] BLUM gives a correct proof of the assertion that speedup is not effective:

THEOREM. [6, th.2]. *Let* R *be sufficiently large and let* f *be total. Then there exists no partial recursive function* σ *such that* σ *converges on indices* i *for* f *yielding indices* $\sigma(i)$ *of* R*-faster programs for* f.

BLUM's proof consists of a machine-dependent argument on Turing machines, which transfers to the general case by recursive relatedness. A machine-independent proof, which uses the double recursion theorem of SMULLYAN, can be found in SCHNORR [22].

The next question is whether the size (i.e., the index) of the faster program can be bounded in terms of the original program, i.e., does there exist a program φ_b converging on indices i of f such that there always exists an index j of an R-faster program for f with $j \leq \varphi_b(i)$.

We have observed above that the faster program for the function f having speedup, as constructed in section 3, can be selected from some well determined finite set and in these circumstances such a bounding function φ_b can be shown to exist. The same holds for functions f having operator speedup, constructed by a similar method (cf. MEYER & FISCHER [17]).

Therefore, people have looked for modifications of the speedup construction in which it becomes nevertheless impossible to bound the size of the faster program.

An attempt of such a modification is given by CONSTABLE & HARTMANIS [8] for the case of functions having R-speedup; their proof, however, seems to be incomplete. A convincing proof has been given by HELM & YOUNG [14], but they need a function having operator speedup.

THEOREM. *Let* R > I. *Then there exists a function* f *such that:*

1) f *has "strong"* R *speedup: If* φ_i = f *there exists an index* j *for* f *such that* $\lambda x[R(\sum_{y\leq x+1} \Phi_j(y))] \propto \Phi_i$,
2) *the size of a shift-faster program cannot be bounded:*
$\exists_b[\forall_i[\varphi_i=f \Rightarrow \exists j[\varphi_j=f$ and $j \leq \varphi_b(i) < \infty$ and $\lambda x[\Phi_j(x+1)] \propto \Phi_i]]]$.

A more general version of this result can be found in MEYER & FISCHER [17, th.5] The question whether the size of the speedup can be bounded for composition-speedup is still as far as I know unsolved.

An argument used to defend the position that speedup is not relevant for practical computing, points to the use of almost everywhere quantifiers in the formulation of the theorem. A program φ_j which is faster than φ_i but only becomes so at arguments beyond 10^{1000} is worthless as far as practical computing is concerned..

Inspecting the different possibilities of encoding the needed finite information in the parameter v, we can make the following observations:

1) using the original strategy of encoding a complete table of function values in v we produce a program which has an efficient run-time at practically all arguments; however, its size is unpredictable.
2) Using the alternative strategy of encoding in v the number of indices < u eventually to be cancelled yields on the other hand a program of bounded size, which is, however, inefficient on an initial segment of unknown length.

Reading these observations one gets the impression that some "inherent trade-off" phenomenon prevents the existence of a program which has both a bounded size and a nice run-time behaviour. That this impression is correct can be concluded from the following result:

THEOREM. (SCHNORR [20,22]). *For sufficiently large total* R *there exists no pair of recursive functions* f *and* h *satisfying:*

1) f *is a total function having* R*-speedup,*
2) h *converges on indices* i *for* f *such that*

$\exists j < h(i)[\varphi_j = f$ and $R \circ \Phi_j \propto \Phi_i$ and $\#\{x \mid R(\Phi_j(x)) > \Phi_i(x)\} \leq h(i)]$
(where # B denotes the number of elements in a set B).

6. SPEEDUP ON INFINITELY MANY ARGUMENTS AND PSEUDO-SPEEDUP

Till now we have considered speedups where the faster program is faster at almost all arguments, and still computes the original function. By requiring less one can define modifications of speedup, which, however, can be shown to be effective under certain circumstances.

In the first place one can weaken the condition on the computed function. A program φ_j is called an R-*pseudo-speedup* of φ_i provided $R \circ \Phi_j \propto \Phi_i$ and $\overset{\infty}{\forall} x[\varphi_i(x) = \varphi_j(x)]$. Examples of pseudo-speedups are the programs $\varphi_{\sigma(i,u,v)}$ for $\varphi_{\sigma(i,0,0)}$ (regardless the value of v).

BLUM has proved that pseudo-speedups can be found effectively whenever they exist.

THEOREM. (BLUM [6, th.5]). *There exists a total function* H *and a transformation* τ *such that the assumptions:*
1) $\varphi_e = R$ *is total and increasing and* $R > H$
2) f *is total and all its programs have* R-*pseudo-speedup*
are sufficient to prove that τ *computes this pseudo-speedup: if* $f = \varphi_i$ *then* $\varphi_{\tau(e,i)}$ *is an* R-*pseudo-speedup of* φ_i.

The idea behind the proof is that the program $\varphi_{\tau(e,i)}$ can seek for a more efficient program for φ_i (which is assumed to exist); at the time the more efficient program is found only a finite number of incorrect values will have appeared.

A more interesting generalization of the speedup concept results from requesting the speedup to be more efficient only a infinitely many arguments.

If the program φ_j computes the same function f as φ_i and satisfies the relation $\overset{\infty}{\exists} x[R(\Phi_j(x)) < \Phi_i(x)]$, then we say that φ_j is an *i.o.* R-*speedup* of φ_i. The fact that φ_j is infinitely often faster than φ_i does not exclude the possibility that φ_j is much slower than φ_i on the rest of its domain. By replacing the program φ_j by a new one which runs φ_i and φ_j in parallel one can produce an i.o. speedup of φ_i which is nowhere much worse than φ_j. By requiring the parallel computation axiom of LANDWEBER & ROBERTSON [15] to be valid, one ensures that the parallel machine is never slower than the original one.

BLUM [6] considered also a type of i.o. speedup by an absolute amount. Let g and r be two total functions with $g >> r$ and let $\varphi_i = \varphi_j = f$. The program φ_j is called a g-r-*levelling* of φ_i provided $\overset{\infty}{\exists} x[\Phi_i(x) > g(x)$ and $\Phi_j(x) < r(x)]$. This condition means that φ_j is infinitely often very efficient at arguments where φ_i is very slow; examples of levellings can be found in BLUM [6].

More interesting are functions with the properties that they have g-r levellings resp. i.o. R-speedup for arbitrarily large g resp. R. These functions have been investigated by BLUM & MARQUES [7]. Their results are formulated in terms of i.o. speedup and levelability of recursively enumerable sets. With each recursively enumerable set A we let correspond its semi-characteristic function χ_A defined by $\chi_A(x) =$ $=$ if $x \in A$ then 1 else ∞ fi. A set A has a certain speedup property provided its semi-characteristic function has it. In order to understand their results we need a few definitions.

A (partial) function f is called *effectively speedable* if there exists a transformation τ satisfying:

$$\forall_{i,\ell}[\text{if } \varphi_\ell = R \text{ is total and } \varphi_i = f \text{ then } \varphi_{\tau(i,\ell)} \text{ is an i.o. R-speedup of } \varphi_i \text{ satisfying } \Phi_{\tau(i,\ell)} \leq \Phi_i].$$

If the above relation holds for some non-recursive function τ the function f is called *speedable* Clearly, a function f is non-speedable if it has for some total function R an R-optimal program.

Analogously a function f is called *effectively levelable* if there exists a transformation τ and a total function r such that

$$\forall_{i,\ell}[\text{if } \varphi_\ell = g > r \text{ is total and } \varphi_i = f \text{ then } \varphi_{\tau(i,\ell)} \text{ is a g-r levelling of } \varphi_i \text{ satisfying } \Phi_{\tau(i,\ell)} \leq \Phi_i].$$

Again the function f is *levelable* if the above relation holds for some non-recursive function τ. Hence a function f is non-levelable if for each total function r one has an index i for f and a total function g > r such that $f = \varphi_j$ implies $\overset{\infty}{\forall} x[\Phi_j(x) < r(x) \Rightarrow \Phi_i(x) < g(x)]$.

A function f is called *effectively non-levelable* if the following (much stronger) condition holds:

Given an index ℓ for r and an index i for f, one can compute an index for a total function g such that the above relation $\overset{\infty}{\forall} x[\Phi_j(x) < r(x) \Rightarrow \Phi_i(x) < g(x)]$ holds for all programs φ_j for f.

From recursive function theory we need the concept of creativity. A recursively enumerable set A is called *creative* if there exists a function t such that:

$$\forall i[t(i) \in (A \cap \mathcal{D}\varphi_i) \cup (\mathbb{N} \setminus (A \cup \mathcal{D}\varphi_i))].$$

Actually the above defines what ROGERS calls *complete creativity*, but the two concepts can be shown to be equivalent (cf. ROGERS [19]). Taking i to be an index such

that $\mathcal{D}\varphi_i \cap A = \emptyset$ one easily sees that $t(i) \in \mathbb{N}\backslash(A \cup \mathcal{D}\varphi_i)$ from which one derives that $\mathbb{N}\backslash A$ is not recursively enumerable.

BLUM & MARQUES use the following generalization of creativity: A recursively enumerable set A is called *subcreative* if there exists a transformation τ such that $\mathcal{D}\varphi_{\tau(i)} = \mathcal{D}\varphi_i \cup \{x_i\}$ where $x_i \in (A \cap \mathcal{D}\varphi_i) \cup (\mathbb{N}\backslash A \cup \mathcal{D}\varphi_i))$.

The difference is that not the element x_i itself is given but only the set which results by adding x_i to $\mathcal{D}\varphi_i$. Again a subcreative set has a complement which is not recursively enumerable.

Using the above definitions we can summarize the main results of BLUM & MARQUES [7] as follows:

THEOREM.

1) *A set* A *is effectively speedable iff it is subcreative; a partial function* f *is effectively speedable iff its graph is subcreative.*
2) *A set* A *is non-speedable if there exists a transformation* σ *such that:*

$\forall i[\mathcal{D}\varphi_i \cap (\mathbb{N}\backslash A) = \mathcal{D}\varphi_{\sigma(i)} \cap (\mathbb{N}\backslash A))$ and $(\mathcal{D}\varphi_i \subset A \Rightarrow \mathcal{D}\varphi_{\sigma(i)}$ is finite)].

3) *A set* A *is levelable iff there exists a transformation* σ *satisfying: for all* i *the program* $\varphi_{\sigma(i)}$ *computes the characteristic function of a recursive set* C_i *and for each recursive set* R *containing the complement of* A *there exists an index* i *such that* $C_i \cap R$ *is infinite and* $C_i \subset A$.
4) *Levelable sets are speedable; there exists, however, a set which is both effectively speedable and effectively non-levelable.*

In BLUM [6, th.4] it has been shown that creative sets are effectively levelable. BLUM & MARQUES give no "if and only if" characterization for effective levelability. The criteria in 2) and 3) seem to have been invented ad hoc; the corresponding concepts in recursion theory have not yet been named. The proofs of the above results are highly complicated.

In section 8 we shall consider the importance of levelability at the bottom of the complexity hierarchy.

7. THE RELATION WITH SPEEDUP IN LOGIC

In 1936 GÖDEL [11] remarks that by extending a theory by an undecidable sentence one does not only get new theorems but also much shorter proofs for some theorems which were already provable. The result can be found also in MOSTOWSKI [18] or EHRENFEUCHT & MYCIELSKI [9], and ARBIB [3,4].

The conditions enforced on the extension to provide the shorter proofs are formulated in the terminology of logic. GÖDEL considers the extension $S_i \subset S_{i+1}$ where S_i is logic of the i-th order (i.e., quantification over objects upto type i). EHRENFEUCHT & MYCIELSKI treat the extension $T \subset T+\alpha$ where it is assumed that $T + \underline{\text{not}}\alpha$ is an unde-

cidable theory. ARBIB considers a first order logic S formalizing natural numbers with +, = and < and considers the extension $S \subset S+\alpha$ where α is an undecidable sentence.

It is reasonable to ask whether this speedup in logic is an appearance of the speedup phenomena in recursion theory described in the preceding sections. This question was investigated by ARBIB [3,4] and the answer seems to be negative. Speedup in logic is due to a much simpler phenomenon than the BLUM speedup.

The sentences provable in some theory S form a recursively enumerable set A. We can consider the theory to be a program having this set as its domain and having the length of the shortest proof of some sentence α in the theory S as its run-time at α. Schematically:

Theory S	$\leftrightarrow$	program φ_S
set of theorems $\{\alpha \mid S \vdash \alpha\}$	$\leftrightarrow$	$\mathcal{D}\varphi_S$
length of shortest proof of α	$\leftrightarrow$	$\Phi_S(\alpha)$.

Now let $S \subset S'$ be an extension of theories. Clearly, $\mathcal{D}\varphi_S \subsetneq \mathcal{D}\varphi_{S'}$, so $\varphi_{S'}$ cannot be considered to be a speedup or i.o. speedup or levelling of φ_S; assuming moreover that there are infinitely many more theorems in S' as there were in S it cannot be a pseudo-speedup either. The real origin of the speedup in logic is the following elementary lemma in recursion theory.

LEMMA. *Let* f *and* g *be partial functions such that* $\mathcal{D}f \subset \mathcal{D}g$ *and such that* $\mathcal{D}g \setminus \mathcal{D}f$ *is not recursively enumerable. Then for every pair of programs* $\varphi_i = f$ *and* $\varphi_j = g$ *and for every total function* R *one has:*

$$\overset{\infty}{\exists} x[x \in \mathcal{D}f \text{ and } R(\Phi_j(x)) < \Phi_i(x)].$$

PROOF. Assume, by hypothesis to be contradicted, that $f = \varphi_i$, $g = \varphi_j$ and $\overset{\infty}{\forall} x[x \in \mathcal{D}f \Rightarrow R(\Phi_j(x)) \geq \Phi_i(x)]$. Then

$$\overset{\infty}{\forall} x[x \in \mathcal{D}g \setminus \mathcal{D}f \text{ iff } (x \in \mathcal{D}g \text{ and } \Phi_i(x) > R(\Phi_j(x)))]$$

which shows $\mathcal{D}g \setminus \mathcal{D}f$ to be recursively enumerable; contradiction. □

A more direct analogy of the BLUM speedup in logic can be derived from the results in the preceding section. Most nice undecidable theories have sets of provable theorems which are creative. Consequently those sets are both effectively speedable and effectively levelable. This implies that a theory like arithmetic can be sped up at infinitely many provable sentences by some conservative extension. On the other hand it seems highly unlikely that the sets having a.e. speedup which are constructed by diagonalization can be at the same time (the encoding of) some interesting theory. No result of this type is known to me.

8. THE RELEVANCE OF SPEEDUP FOR DECIDABLE THEORIES

In the preceding section we have seen that any reasonable undecidable theory is speedable; choosing a stronger formalism one produces proofs which are shorter by a certain amount. Since the theory is undecidable, the length of proofs cannot be bounded in advance; therefore, assuming that the theorems whose proofs are shortened originally had proofs of very high complexity, the new theory can still be as prohibitive as the old one. Hence it is not clear that this speedup result concerns the actual practice of mathematics.

If we consider a decidable theory (i.e., the set of provable sentences is recursive) the situation changes. A recursive set is neither speedable nor levelable, but it can have R-speedup resp. g-r levelling for functions R resp. r which are small compared to the actual complexity of the set. The question arises whether any of these phenomena will occur in the situation of some nice decidable theory.

Recent work by MEYER, STOCKMEYER [23] and others on the complexity of decidable theories show that interesting theories have strong levelability properties. These results are obtained not so much by direct construction, but as a consequence of their techniques of translating Turingmachine computations into theories about regular expressions, orderings, arithmetic etc.

In the discussion below the domain will be the set Σ^* of strings over some finite alfabet (e.g., {0,1}); the length of a string x is denoted $|x|$.

Let A be a set $A \subset \Sigma^*$ such that A can be recognized by a Turing machine M such that the tape used by M on a string x is bounded by an exponential function in $|x|$, say $2^{|x|}$. This is expressed by writing $A \in$ EXPSPACE. In the sequel it does not matter whether M works deterministically or not. Clearly, $x \in A$ iff the string $\not\in \times b^k \not\in$ where $k = 2^{|x|} - |x|$, b denotes the blank symbol of M and $\not\in$ an endmarker, is the first instantaneous description (i.d.) of an accepting computation of M which never leaves the tape presented in the first i.d.

On the other hand we can consider regular expressions over the alfabet Σ, formed using the operators $\cup$ (union), $\cdot$ (concatenation), * (Kleene star), and ^{2} (squaring); these expressions form the set $REG(\Sigma,\{\cup,\cdot,^*,^2\})$. Such a regular expression describes a regular subset of Σ^* which can be either the complete set Σ^* or not. It is decidable whether the expression describes the full set Σ^* and consequently the set $NEC(\Sigma,\{\cup,\cdot,^*,^2\})$ of all regular expressions describing regular sets having non-empty complement is recursive; actually it is a member of EXPSPACE.

The main technique of MEYER & STOCKMEYER consists of the construction of mappings like the mapping f below:

LEMMA. (STOCKMEYER [23]). *There exists a function* $f: \Sigma^* \to REG(\Sigma,\{\cup,\cdot,^*,^2\})$ *such that:*

1) f *is computable in* log *space, stretching (i.e.* $\forall x[|x| \leq |f(x)| \leq K\cdot|x|]$ *for some constant* K*), and* log*-space invertible* (a technical condition needed to use it for reducibilities).

2) *For each* x *the expression* f(x) *describes a regular set consisting of all strings except (if present) the encodings of accepting computations of* M *on* x *on* $2^{|x|}$ *tape squares.*

Consequently one has $x \in A$ iff $f(x) \in NEC(\Sigma,\{\cup,\cdot,^*,^2\})$. From this one derives that the set $NEC(\Sigma,\{\cup,\cdot,^*,^2\})$ is complete in EXPSPACE, which was the main goal of the technique.

However, the technique proves at the same time that the set $NEC(\Sigma,\{\cup,\cdot,^*,^2\})$ has a strong levelability property. Starting point is the following result:

LEMMA. (MEYER & STOCKMEYER [23, fact 3.11]). *Let* S > log *be tape constructable and let* R = o(S) *then there exists a set* A, *which is recognized deterministically in space* S *such that* A *is effectively* R-Z-*levelable* (remember that $Z = \lambda x[0]$).

The following proof of this result (which is to short to be omitted) was communicated to me by HARTMANIS [12].

PROOF. Let A be the set of all strings of the form x # w such that the Turing machine M_x with index x on input w either uses more than $S(|x\#w|)/|x|$ tape squares, or uses less than $S(|x\#w|)/|x|$ tape squares but rejects its input. Assuming the Gödel numbering to be reasonable, the computation of M_x on k tape squares can be simulated by some universal machine on $|x|\cdot k$ squares. Hence the set A can be recognized in space S.

However, if M_y is a machine accepting A one derives that the regular set $\{y\#w \mid w \in \Sigma^*\}$ is entirely contained within A, and moreover M_y takes more than $S(|y\#w|)/|y|$ tape squares to accept the string y # w. Consequently a new machine which first (by finite automaton action) tests whether the input starts with y#, and otherwise simulates M_y on the input, provides an effective R-Z levelling of A. □

From this lemma we conclude that there exists a set A recognizable in exponential space, which is exponential-to-zero levelable. The next lemma shows that this levelability is preserved under reducibilities:

LEMMA. *Let* A *be* S-log-*levelable and let* $A \leq B$ *by* f *where* f *is stretching and* log-*space invertible. Then there exists a constant* c > 0 *such that* B *is* $\lambda x[S(cx)]$-log-*levelable.*

The consequence of these lemmas becomes clear. There exists some weird set A in EXPSPACE which is exp-log-levelable, but since this set can be reduced to the set $NEC(\Sigma,\{\cup,\cdot,^*,^2\})$ by a function f having the right properties, the latter set is exp-log-levelable as well. But $NEC(\Sigma,\{\cup,\cdot,^*,^2\})$ is an interesting mathematical object.

Encodings like the mapping f described above have been constructed for many interesting decidable theories; a well known example is Pressburger arithmetic, for which FISCHER & RABIN [10] have proved a two-fold exponential lower time bound. Other theories like the weak monadic second order theory of successor have been shown to require a non-elementary amount of time. (cf. MEYER [16]). Each of the above results

yields as a corollary a corresponding strong levelability result on the theory under discussion.

The above shows us something new about the doom of formalism. It is not only the case that each formalism prevents us from proving some interesting valid theorem, but, even when we consider its restriction to some non-trivial but decidable theory, there will be sequences of trivialities, recognizable (e.g., by some finite automaton) for which the formalism itself only provides prohibitive proofs.

REFERENCES

1. ALTON, D., *Diversity of Speedups and embeddability in computational complexity*, Rep. TR 73-01, Dept. Comp. Sci., Univ. of Iowa (Iowa City, 1973).

2. ALTON, D., *Non-existence of program optimizers in an abstract setting*, Rep. TR 73-08, Dept. Comp. Sci., Univ. of Iowa (Iowa City, 1973).

3. ARBIB, M.A., *Speedup theorems and Incompleteness theorems*, in Automata theory, (ed. E.R. Caianello), pp.6-24, Academic Press, New York, 1966.

4. ARBIB, M.A., *Speedup and incompleteness results*, in Theories of abstract automata, pp.261-267, Prentice Hall, New Jersey, 1969.

5. BLUM, M., *A machine-independent theory of the complexity of recursive functions*, J. Assoc. Comput. Mach., 14(1967), 322-336.

6. BLUM, M., *On effective procedures for speeding up algorithms*, J. Assoc. Comput. Mach. 18(1971), 290-305.

7. BLUM, M. & I. MARQUES, *On complexity properties of recursively enumerable sets*, J. Symbolic Logic, 38(1973), 579-593.

8. CONSTABLE, R.L; & J. HARTMANIS, *Complexity of formal translations and speedup results*, in proc. 3rd ACM Symp. on theory of Computing (1971), pp.244-250.

9. EHRENFEUCHT, A. & J. MYCIELSKI, *Abbreviating proofs by adding new axioms*, Bull. Amer. Math. Soc, 77(1971), 366-367.

10. FISCHER, M.J. & M.O. RABIN, *Super-exponential complexity of Pressburger arithmetic*, MAC techn. memo 43, Project MAC. MIT Cambridge Mass. Feb. 1974.

11. GÖDEL, K., *Über die Länge der Beweise*. Ergebnisse eines Math. Kolloquiums, 7, 23-24 (1936). See also in *On the lengths of proofs* in the Undecidable, (ed. M. Davis), pp.82-83, Raven press, NY, 1965.

12. HARTMANIS, J., Oral communication, Dec. 1974.

13. HARTMANIS, J. & J.E. HOPCROFT, *An overview of the theory of computational complexity*, J. Assoc. Comput. Mach. 18(1971), 444-475.

14. HELM, J.P. & P.R. YOUNG, *On size versus efficiency for programs admitting speedup*, J. Symbolic logic, 36(1971), 21-27.

15. LANDWEBER, L.H. & E.L. ROBERTSON, *Recursive properties of abstract complexity classes*, J. Assoc. Comput. Mach., 19(1972), 296-308.

16. MEYER, A.R., *Weak monadic second order theory of successor is not elementary recursive*, MAC techn. memo 38. Project MAC, MIT, Cambridge Mass. 1973.

17. MEYER, A.R. & P.C. FISCHER, *Computational speedup by effective operators*, J. Symbolic Logic, 37(1972), 55-68.

18. MOSTOWSKI, A., *Sentences undecidable in formalized arithmetic*, North Holland, Amsterdam, 1957.

19. ROGERS, H., *The theory of recursive functions and effective computability*, Mc Graw Hill, New York, 1967.

20. SCHNORR, C.P., *Does computational speedup concern programming?*, in Automata Languages and Programming, (M. Nivat ed.), pp.585-592, North Holland/ Elsevier, Amsterdam, 1973.

21. SCHNORR, C.P. & G. STUMPF, *A characterization of complexity sequences*, Tagungsbericht 46/1972, Algorithmen und Komplexitätstheorie, Math. Forschungsinstitut Oberwolfach, Nov. 1972.

22. SCHNORR, C.P., *Rekursive Funktionen und ihre Komplexität*, Teubner, Stuttgart, 1974.

23. STOCKMEYER, L.J., *The complexity of decision problems in Automata theory and Logic*, Report MAC TR-133, Project MAC. MIT, Cambridge Mass., July 1974.

24. YOUNG, P., *Speedups by changing the order in which sets are enumerated*, Math. Systems theory, 5(1971), 148-156.

25. YOUNG, P., *Easy constructions in complexity theory: Gap and Speedup theorems*, Proc. Amer. Math. Soc., 37(1973), 555-563.