

Speed-up Theorems in Type-2 Computation

Chung-Chih Li

School of Information Technology
Illinois State University, Normal, IL 61790-5150, USA
cli2@ilstu.edu

Abstract. A classic result known as the speed-up theorem in machine-independent complexity theory shows that there exist some computable functions that do not have best programs for them [2, 3]. In this paper we lift this result into type-2 computation under the notion of our type-2 complexity theory depicted in [15, 13, 14]. While the speed-up phenomenon is essentially inherited from type-1 computation, we cannot directly apply the original proof to our type-2 speed-up theorem because the oracle queries can interfere the speed of the programs and hence the cancellation strategy used in the original proof is no longer correct at type-2. We also argue that a type-2 analog of the operator speed-up theorem [16] does not hold, which suggests that this curious phenomenon disappears in higher-typed computation beyond type-2.

1 Introduction

Speed-up phenomena have been extensively studied by mathematicians for more than a half century, which was first remarked by Gödel [8] in the context of the theorem proving.¹ In [2, 3] Blum re-discovered the speed-up theorem in terms of computable functions and his complexity measures. The theorem asserts that the best program does not always exist for some computable functions. In order to state the theorem precisely, we first fix some notations and conventions. By computable we mean Turing machine computable. A function is said to be recursive if it is total and computable. Let φ_e denote the function computed by the e^{th} Turing machine and Φ_e denote the cost function associated to the e^{th} Turing machine. More precisely, let $\langle \varphi_i \rangle_{i \in \mathbf{N}}$ be an *acceptable programming system* [17] and $\langle \Phi_i \rangle_{i \in \mathbf{N}}$ be a *complexity measure* [2] associated to $\langle \varphi_i \rangle_{i \in \mathbf{N}}$, where \mathbf{N} is the set of natural numbers. The standard asymptotic notion, $\overset{\infty}{\forall}$, is read as *for all but finitely many*². We state the original speed-up theorem as follows:

Theorem 1 (The Speed-up Theorem [2, 3]). *For any recursive function r , there exists a recursive function f such that*

$$(\forall i : \varphi_i = f) (\exists j : \varphi_j = f) (\overset{\infty}{\forall} x) [r(\Phi_j(x)) \leq \Phi_i(x)].$$

¹ The original remarks were translated in [7], pages 82-83. More discussion about the relation between the computational speed-up phenomena and Gödel's speed-up results in logic can be found in [21].

² The negation of “for all but finitely many” is “exist infinitely many” denoted by $\overset{\infty}{\exists}$.

We say that function f in the theorem above is r -speedable. We revise the original proof of this theorem so that it can be easily modified for our type-2 speed-up theorem (see [12] for details). More original proofs can be found in [2, 3, 21, 6, 22, 4, 19]. Many variations of the speed-up theorem have been proven since Blum's [2, 3]. We are interested in Meyer and Fischer's operator speed-up theorem [16] where the speed-up factor r , a recursive function, is strengthened to an effective operator Θ as follows:

Theorem 2 (The Operator Speed-up Theorem [16]). *For any total effective operator Θ , there is a recursive function f that can be uniformly constructed such that*

$$\forall i : \varphi_i = f \exists j : \varphi_j = f \bigvee_{x \in \mathbb{N}} x[\Theta(\Phi_j)(x) \leq \Phi_i(x)].$$

Our goal in the present paper is to lift these two speed-up theorems to type-2 computation. We obtain a type-2 analog of Theorem 1. However, Theorem 2 fails to hold in the context of type-2 computation, which suggests that there always exist best programs in higher-typed computation beyond type-2.

In the next section, we briefly introduce the current status of type-2 complexity theory and describe some necessary preliminaries. These paragraphs are perforce brief and superficial due to space constraints. Since the speed-up theorem is for the most part independent from the other parts of the theory, our coverage will be limited to the topics pertinent to our results. More details can be found in [15, 13, 14].

2 Conventions & Type-2 Complexity Theory

We consider natural numbers as type-0 objects and functions over natural numbers as type-1 objects. Type-2 objects are called *functionals* that take as inputs and produce as outputs type-0 or type-1 objects. By convention, we consider objects of lower type as special cases of higher type, and thus, type-0 \subset type-1 \subset type-2. Without loss of generality we restrict type-2 functionals to our standard type $\mathcal{T} \times \mathbb{N} \rightarrow \mathbb{N}$, where \mathcal{T} is the set of total functions and \rightarrow means possibly partial. Note that $f \in \mathcal{T}$ may not be computable. For $n \in \mathbb{N}$, $|n|$ denotes the length of the binary bit string representing n . For type-2 computation we use the Oracle Turing Machine (OTM) as our standard computing formalism. An OTM is a Turing machine equipped with a function oracle. Before an OTM begins to run, the type-1 argument should be presented to the OTM as an oracle. In addition to the standard I/O tape for type-0 input/output and intermediate working space, an OTM has two extra tapes – one is for oracle queries and the other one is for the answers to the queries. During the course of the computation, the OTM may enter a special state called query-state. In this state the oracle will read the query left on the query-tape and prepare its answer on the the answer-tape for the OTM to read. All this will be done at no cost to the OTM. However, the OTM has to prepare the queries and read their answers at its own computational cost. We also fix a programming system $\langle \hat{\varphi}_i \rangle_{i \in \mathbb{N}}$ associated with

some complexity measure $\langle \widehat{\Phi}_i \rangle_{i \in \mathbf{N}}$ for OTM. By convention, we take the number of steps as our time complexity measure, i.e., the number of times an OTM moves its read/write heads. Also, we use \widehat{M}_e to denote the OTM with index e and $\widehat{\varphi}_e$ is the functional computed by \widehat{M}_e . Following these conventions, Seth [20] adapted Hartmanis and Stearns's notion [9] to define type-2 complexity classes. He proposed two alternatives:

1. Given recursive $t : \mathbf{N} \rightarrow \mathbf{N}$, let $DTIME(t)$ denote the set of type-2 functionals such that, for every functional $F \in DTIME(t)$, F is total and there is an OTM \widehat{M}_e that computes F and, on every $(f, x) \in \mathcal{T} \times \mathbf{N}$, \widehat{M}_e halts within $t(m)$ steps, where $m = |\max(\{x\} \cup Q)|$ and Q is the set of all answers returned from the oracle during the course of the computation.
2. Given computable functional $H : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$, let $DTIME(H)$ denote the set of type-2 functionals such that, for every functional $F \in DTIME(H)$, F is total and there is an OTM \widehat{M}_e that computes F and, on every $(f, x) \in \mathcal{T} \times \mathbf{N}$, \widehat{M}_e halts within $H(f, x)$ steps.

The key idea behind Seth's complexity classes is directly lifted from [9]. The same machine characterization idea can also be found in other works such as Kapron and Cook's [10] and Royer's [18]. In Seth's first definition stated above, the resource bound is determined by the sizes of oracle answers; but the set Q in the definition of $DTIME(t)$ in general is not computable and hence can't be available before the computation halts, if ever. Alternatively, we may update the bound dynamically upon each answer returned from the oracle during the course of the computation. But if we do so, there is no guarantee that a clocked OTM must be total. For example, Cook's POTM [5] is an OTM bounded by a polynomial in this manner but a POTM may run forever. Kapron and Cook's proposed their remedies in the context of feasible functionals and gave a very neat characterizations of type-2 Basic Feasible Functionals (BFF) in [10], where the so-called second-ordered polynomial is used as the bound. In [13, 14] we adapted all these ideas and extended the second-ordered polynomial to a general type-2 computable functional to have the following complexity class:

$$DTIME(H) = \{F \mid \exists e[\widehat{\varphi}_e = F \text{ and } \widehat{\Phi}_e \leq_2^* H]\}. \quad (1)$$

The relation, \leq_2^* , used above will be defined in Definition 3, which is crucial to our works. Along the lines of the classical complexity theory initiated by a series of seminal papers [9, 2, 3], our previous results in [15, 13, 14] show that the complexity theory at type-2 does not parallel to its type-1 counterpart. To begin with, we defined \leq_2^* with a workable and reasonable type-2 analog of asymptotic notion. We equated our notion of *finitely many* at type-2 to the *compact sets* in some Baire-like topology [1] that was relatively defined by the concerned functionals. As there is no type-2 equivalent of Church-Turing thesis, the compactness in our definition is the key to computability of our construction. In [14] we examined some alternative clocking schemes for OTM and defined a class of limit functionals determined by some computable functions to serve as type-2 time bounds. With these type-2 time bounds, we were able to define an explicit

type-2 complexity class similar to (1) for a general type-2 complexity theory. Unlike many other complexity theorems, the speed-up theorems do not need a precisely defined complexity classes. We thus skip details regarding our explicit type-2 complexity classes. However, the asymptotic notion is still indispensable in the present paper. We formalize the notion as follows. Let \mathcal{F} denote the set of *finite* domain functions over natural numbers, i.e., $\sigma \in \mathcal{F}$ iff $\text{dom}(\sigma)$ is finite. Given $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$, let $F(f, x) \downarrow = y$ denote the case that F is defined at (f, x) and its value is y . For $\sigma \in \mathcal{F}$ and $f \in \mathcal{F} \cup \mathcal{T}$, let $\sigma \subset f$ denote the case that f is an extension of σ .

Definition 1. *Let $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$ and $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$. We say that (σ, x) is a locking fragment of F if and only if*

$$\exists y \in \mathbf{N} \forall f \in \mathcal{T} [\sigma \subset f \Rightarrow F(f, x) \downarrow = y].$$

Also, we say that (σ, x) is a *minimal locking fragment* of F if (σ, x) is a locking fragment of F and, for every $\tau \in \mathcal{F}$ with $\tau \subset \sigma$, (τ, x) is not a locking fragment of F . Clearly, if F is total and computable, then for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, there must exist a unique $\sigma \in \mathcal{F}$ with $\sigma \subset f$ such that (σ, x) is a minimal locking fragment of F . It is also clear that, in general, whether or not (σ, x) is a minimal locking fragment of F cannot be effectively decided. For any $\sigma \in \mathcal{F}$, let $((\sigma))$ be the set of total extensions of σ , i.e., $((\sigma)) = \{f \in \mathcal{T} \mid \sigma \subset f\}$. Also, if $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, let $((\sigma, x)) = \{(f, x) \mid f \in ((\sigma))\}$. We observe that, $((\sigma_1)) \cap ((\sigma_2)) = ((\sigma_1 \cup \sigma_2))$ if σ_1 and σ_2 are consistent; otherwise, $((\sigma_1)) \cap ((\sigma_2)) = \emptyset$. The union operation $((\sigma_1)) \cup ((\sigma_2))$ is conventional. Given any $f, g \in \mathcal{T}$, it is clear that, if $f \neq g$, then there exist $\sigma \subset f, \tau \subset g$, and $k \in \text{dom}(\sigma) \cap \text{dom}(\tau)$ such that $\sigma(k) \neq \tau(k)$. In stead of taking every $((\sigma, x))$ with $\sigma \in \mathcal{F}$ as the basic open set³, we consider only those that are related to the concerned functionals as follows.

Definition 2. *Given any continuous functionals, F_1 and F_2 , let $\mathbb{T}(F_1, F_2)$ denote the topology induced from $\mathbb{T} \times \mathbf{N}$ by F_1 and F_2 , where the basic open sets are defined as follows: $((\sigma, a))$ is a basic open set of $\mathbb{T}(F_1, F_2)$ if and only if, for some $(f, a) \in \mathcal{T} \times \mathbf{N}$, (σ_1, a) and (σ_2, a) are the minimal locking fragments of F_1 and F_2 , respectively, and $((\sigma, a)) = ((\sigma_1, a)) \cap ((\sigma_2, a))$.*

Note that, in the definition above, since $((\sigma, a)) = ((\sigma_1, a)) \cap ((\sigma_2, a)) = ((\sigma_1 \cup \sigma_2, a))$ we have that if $((\sigma, a))$ is a basic open set of $\mathbb{T}(F_1, F_2)$, then (σ, a) must be a locking fragment of both F_1 and F_2 . Let $X_{[F_1 \leq F_2]} \subseteq \mathcal{T} \times \mathbf{N}$ denote the set $\{(f, a) \mid F_1(f, a) \leq F_2(f, a)\}$. $X_{[F_1 > F_2]}$ is simply the complement of $X_{[F_1 \leq F_2]}$ called the *exception set* of $F_1 \leq F_2$. Now, we are in a position to define our type-2 almost-everywhere relation.

Definition 3. *Let $F_1, F_2 : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$ be continuous. Define*

$$F_1 \leq_2^* F_2 \text{ if and only if } X_{[F_1 \leq F_2]} \text{ is co-compact in } \mathbb{T}(F_1, F_2).$$

³ This will form the product topology $\mathbb{T} \times \mathbf{N}$, where \mathbb{T} is the Baire topology and \mathbf{N} the discrete topology on \mathbf{N} .

Using the same idea of compactness in Definition 3, two modified quantifiers, for all but finitely many and exist infinitely many, can be understood in type-2 context as follows: For continuous functionals $F, G : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$, we have $\forall_2^\infty (f, x)[F(f, x) \leq G(f, x)]$ if and only if $\{(f, x) \mid F(f, x) \leq G(f, x)\}$ is compact in $\mathbb{T}(F, G)$. Similarly, we say that $\exists_2^\infty (f, x)[F(f, x) \leq G(f, x)]$ if and only if $\{(f, x) \mid F(f, x) \leq G(f, x)\}$ is not compact in $\mathbb{T}(F, G)$. One can verify that

$$F \leq_2^* G \iff \forall_2^\infty (f, x)[F(f, x) \leq G(f, x)] \iff \neg \exists_2^\infty (f, x)[F(f, x) > G(f, x)].$$

When the concerned functionals F and G are clear from the context, we simply read $\forall_2^\infty (f, x)$ as “for all (f, x) except those in a compact set such that”, and $\exists_2^\infty (f, x)$ as “there exists a noncompact set such that, for all (f, x) in the set”, where *compact* is understood as $\mathbb{T}(F, G)$ -compact.

3 Lifting Speed-up Theorems to Type-2

Since type-1 computations are just a special case of type-2 computations, the speedable function constructed for the original speed-up theorem can be seen as a type-2 functional that just does not make any oracle queries. In other words, as long as the concerned complexity measure satisfies Blum’s two axioms, the proof of the original speed-up theorem should remain valid at type-2. Clearly, our standard complexity measure $\langle \widehat{\Phi}_i \rangle$, the number of steps the OTM performs, does satisfy Blum’s two axioms. However, we observe that oracle queries in type-2 computation have introduced some difficulties when we attempt a direct translation of the original proof. Recall that the original construction of the speedable function is based on the cancellation on some programs when their run times fall into certain ranges. When we directly lift the construction to type-2, we note that there are cases in which the oracle queries may be used to slow down or speed up the computation in such a way the programs can escape from being canceled. Note that the proofs of the Union Theorem and Gap Theorem do not involve the cancellation but directly construct time bounds and let the definition of the complexity class take care of the rest. Unfortunately, one can easily show that there are functionals that always make unnecessary oracle queries. Consider functional $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$ defined by,

$$F(f, x) = \begin{cases} f(0) + 1 & \text{if } \varphi_x(x) \downarrow \text{ in } f(0) \text{ steps;} \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Clearly, F is computable and total. Fix any a such that, $\varphi_a(a) \uparrow$. Then, on input (f, a) , the value of $f(0)$ only affects the speed of computing $F(f, a)$. Thus, $F(f, a) = 0$ for any $f \in \mathcal{T}$, and hence (\emptyset, a) is the minimal locking fragment of F on (f, a) . That means any queries made during the computation of F on (f, a) are unnecessary. Thus, if there were an OTM that would not make any unnecessary queries for F , one could modify such OTM to solve the halting problem, which is impossible. However, the answer to the query does affect the

speed of the machine to halt. The smaller the value of $f(0)$ is, the sooner the computation halts. In fact, it is easy to construct computable functionals that make unnecessary queries on all inputs, and moreover, the number of unnecessary queries can be arbitrarily large. Such kind of unnecessary but speed-affecting queries is the problem for us to get around in lifting the speed-up theorems into type-2.

It is clear that our $\widehat{\varphi}$ -programming system for OTM can be used to code the entire class of type-1 computable functions. Thus, the speedable function constructed in the original speed-up theorem can be coded in our $\widehat{\varphi}$ -programming system. To that speedable function, any queries made during the course of computation are unnecessary. However, as we have seen, unnecessary queries may affect the computational time. Therefore, we cannot simply cancel those $\widehat{\varphi}$ -programs that make oracle queries. Moreover, if we intuitively enumerate all possible queries in our construction, we face another difficulty in trying to make our speedable functional total, because we cannot decide whether a query is necessary or not; thus our construction will tend to be fooled by infinitely many unnecessary queries and fail to converge. Fortunately, we will see that our notion of \leq_2^* defined by Definition 3 based on the compactness of the relative topologies (Definition 2) resolves this problem automatically and easily.

4 Type-2 Speed-up Theorems

Type-2 speed-up theorems vary with the nature of the speed-up factors that can be either type-1 or type-2. For type-3 speed-up factors, the theorem becomes a type-2 analog of the operator speed-up theorem, and we will argue that there is no such theorem. From Theorem 2 (the operator speed-up theorem) we immediately have the following corollary, in which we replace the operator $\Theta : \mathcal{T} \rightarrow \mathcal{T}$ by a functional $R : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$.

Corollary 1. *For any computable functional $R : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$, there exists a recursive function f such that,*

$$\forall i : \varphi_i = f \exists j : \varphi_j = f \bigvee_{x \in \mathbf{N}} [R(\Phi_j, x) \leq \Phi_i(x)].$$

However, this corollary is of no interest. Our goal is to construct a type-2 speedable functional using our programming system $\langle \widehat{\varphi}_i \rangle_{i \in \mathbf{N}}$ for OTM. We are interested in the following two theorems:

Theorem 3. *For any recursive function $r : \mathbf{N} \rightarrow \mathbf{N}$, there exists a computable functional $F_r : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$ such that,*

$$\forall i : \widehat{\varphi}_i = F_r \exists j : \widehat{\varphi}_j = F_r [r \circ \widehat{\Phi}_j \leq_2^* \widehat{\Phi}_i].$$

Theorem 4 (Type-2 Speed-up Theorem). *For any computable functional $R : \mathcal{T} \times \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$, there exists a computable functional $F_R : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$ such that,*

$$\forall i : \widehat{\varphi}_i = F_R \exists j : \widehat{\varphi}_j = F_R [\lambda f, x. R(f, x, \widehat{\Phi}_j(f, x)) \leq_2^* \widehat{\Phi}_i].$$

Theorem 3 and Theorem 4 are obtained by lifting Theorem 1 and Theorem 2, respectively, into type-2 computation. Note that since Theorem 3 is a special case of Theorem 4, we rather consider Theorem 4 as our type-2 speed-up theorem. Instead of proving Theorem 4 directly, we prove a simpler result of Theorem 3. The idea can be applied to prove Theorem 4.

Consider Theorem 3. We observe that $F_r = \widehat{\varphi}_i = \widehat{\varphi}_j$. By Definition 3, the relative topology for the type-2 relation, $r \circ \widehat{\Phi}_j \leq_2^* \widehat{\Phi}_i$, is

$$\mathbb{T}(r \circ \widehat{\Phi}_j, \widehat{\Phi}_i) = \mathbb{T}(r \circ \widehat{\varphi}_j, \widehat{\varphi}_j) = \mathbb{T}(\widehat{\varphi}_j) = \mathbb{T}(\widehat{\varphi}_i) = \mathbb{T}(F_r).$$

Thus, if we construct F_r with (\emptyset, x) as its minimal locking fragment for every $x \in \mathbf{N}$, then the relative topology for \leq_2^* in the theorem is the coarsest one, i.e., the topology with basic open sets: $((\emptyset, 0)), ((\emptyset, 1)), \dots$. Our idea is that: given any $S \subset \mathcal{T} \times \mathbf{N}$ with S being noncompact in the topology $\mathbb{T}(F_r)$, we then must have that the type-0 component of the elements of S has infinitely many different values. If a $\widehat{\varphi}$ -program i needs to be canceled, we thus have infinitely many chances to do so on some type-0 inputs. We can therefore ignore the effects of the type-1 input in the computation. In other words, it is not necessary to introduce another parameter for the type-1 argument when defining the cancellation sets.

This wishful thinking, however, is problematic in the corresponding type-2 pseudo-speed-up theorem. Because, for every $\widehat{\varphi}$ -program i for F_r , its *pseudo* sped-up version, $\widehat{\varphi}$ -program j , does not exactly compute $\widehat{\varphi}_i$ on some finitely many type-0 inputs, and hence $\widehat{\varphi}_i$ and $\widehat{\varphi}_j$ may define two different topologies. Thus, if we ignore the effect of the type-1 argument, the almost everywhere relation $r \circ \widehat{\Phi}_j \leq_2^* \widehat{\Phi}_i$ may fail in topology $\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_j)$. To fix this problem, we introduce a weaker type-2 pseudo-speed-up theorem, in which the compactness is not considered. The theorem is weaker in a sense that we do not use the type-2 almost everywhere relation. Nevertheless, this weaker type-2 pseudo-speed-up theorem will be sufficient for our proof of Theorem 3.

Theorem 5 (Type-2 Pseudo-Speed-up Theorem). *For any recursive function function $r : \mathbf{N} \rightarrow \mathbf{N}$, there exists a computable functional $F_r : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$ such that, for every $\widehat{\varphi}$ -program i for F_r , there is another $\widehat{\varphi}$ -program j such that,*

$$\forall^\infty x \in \mathbf{N} \forall f \in \mathcal{T} [(\widehat{\varphi}_j(f, x) = F_r(f, x)) \wedge (\widehat{\Phi}_i(f, x) > r \circ \widehat{\Phi}_j(f, x))].$$

We shall omit detailed proof of this pseudo-speed-up theorem due to space constraints and refer readers to [12] for details.

Proof of Theorem 3: According to the construction of $\widehat{\varphi}_e$ in Theorem 5, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{\varphi}_e(0, f, x) = \widehat{\varphi}_e(0, f_0, x)$, where $f_0 = \lambda x.0$. It follows that (\emptyset, x) is the minimal locking fragment of $\widehat{\varphi}_{s(e,0)}$ on every $(f, x) \in \mathcal{T} \times \mathbf{N}$. Let $\widehat{\varphi}_i = \widehat{\varphi}_{s(e,0)}$ and $j = s(e, i + 1)$. Note that $\widehat{\varphi}_i =_2^* \widehat{\varphi}_j$ and $r \circ \widehat{\Phi}_j \leq_2^* \widehat{\Phi}_i$ does not hold in general because $((\emptyset, x))$ may not be a basic open set for some x . Consider the following exception set

$$E = \{(f, x) \mid \widehat{\varphi}_i(f, x) \neq \widehat{\varphi}_j(f, x)\}.$$

Although E may not be compact in topology $\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_j)$, $\{x|(f, x) \in E\}$ must be finite. Thus, we can have a patched $\widehat{\varphi}$ -program j' such that the program will search a look-up table if the type-0 argument is in $\{x|(f, x) \in E\}$. In such a way, the type-1 input will not affect the result, and hence the minimal locking fragment becomes (\emptyset, x) . On the other hand, if type-0 argument $x \notin \{x|(f, x) \in E\}$, then $\widehat{\varphi}$ -program j' starts running $\widehat{\varphi}$ -program j . Similarly, consider

$$E' = \left\{ (f, x) \mid r \circ \widehat{\Phi}_{j'}(f, x) > \widehat{\Phi}_i(f, x) \right\}.$$

Set $\{x|(f, x) \in E'\}$ is finite. Also, consider the patched $\widehat{\varphi}$ -program, j' . We have

$$E'' = \left\{ (f, x) \mid r \circ \widehat{\Phi}_{j'}(f, x) > \widehat{\Phi}_i(f, x) \right\},$$

and $\{x|(f, x) \in E''\}$ is finite. Therefore, E'' is compact in $\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_{j'})$, because $\widehat{\varphi}_i = \widehat{\varphi}_{j'}$ and, for every $x \in \mathbf{N}$, (\emptyset, x) is the only basic open set in $\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_{j'})$.

Finally, we shall discuss the case that there may exist some best $\widehat{\varphi}$ -program for $\widehat{\varphi}_{s(e,0)}$ using some unnecessary queries to escape from being canceled. This is possible because we replace the actual type-1 input by f_0 for every $\widehat{\varphi}$ -program, and hence we do not know the program's behavior on actual $f \in \mathcal{T}$. Clearly, by Claim 6 in the proof of Theorem 5, this problem can be ignored, because any program that will make any query on some inputs does not compute our speedable functional. This completes the proof of Theorem 3. \square

5 Type-2 Operator Anti-speed-up Theorem

In the previous section we established two speed-up theorems. The speed-up factor in Theorem 3 is a type-1 function and the proof is directly modified from a proof for the original speed-up theorem. In Theorem 4 we consider type-2 speed-up factors and it is lifted from the original operator speed-up theorem. In this section we consider a type-2 analog of the operator speed-up theorem, namely, we will try to explore a speed-up phenomenon when the speed-up factor is type-3. Clearly, a proof for such theorem needs a general type-2 s-m-n and a type-2 recursion theorem, which we don't have. Instead, we argue that the type-2 analog of the operator speed-up theorem does not exist.

By "an effective type-2 operator" we mean a computable type-3 functional [11] of type $(\mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}) \rightarrow (\mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N})$ with inputs restricted to computable total type-2 functionals. Thus, we can think up that an effective type-2 operator is computed by a $\widehat{\varphi}$ -program that takes a total $\widehat{\varphi}$ -program as its input and outputs another total $\widehat{\varphi}$ -program. Our next theorem asserts that there is an effective type-2 operator $\widehat{\Theta}$ such that, for *every* total $\widehat{\varphi}$ -program e , there is no $\widehat{\Theta}$ -sped up version for e . In other words, the $\widehat{\Theta}$ -best programs always exist. Our theorem is stronger than a direct negation of the operator speed-up theorem in the sense that we claim that every $\widehat{\varphi}$ -program is a $\widehat{\Theta}$ -best $\widehat{\varphi}$ -program.

Theorem 6 (Type-2 Operator Anti-Speed-up Theorem). *There is a type-2 effective operator $\widehat{\Theta} : (\mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}) \rightarrow (\mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N})$ such that, for every*

computable functional, $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$, we have

$$\forall i : \widehat{\varphi}_i = F \forall j : \widehat{\varphi}_j = F \overset{\infty}{\exists}_2 (f, x) [\widehat{\Theta}(\widehat{\Phi}_j)(f, x) > \widehat{\Phi}_i(f, x)].$$

Proof: Define $\widehat{\Theta} : (\mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}) \rightarrow (\mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N})$ by

$$\widehat{\Theta}(F)(f, x) = f(2^{F(f, x)+1}).$$

Clearly, such $\widehat{\Theta}$ is a type-2 effective operator. Fix any computable $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$. Also, fix a $\widehat{\varphi}$ -program i for F . By contradiction, suppose that j is a $\widehat{\Theta}$ -sped-up version of i , i.e., $\widehat{\Theta}(\widehat{\Phi}_j) \leq_2^* \widehat{\Phi}_i$. If so, for all but finitely many $x \in \mathbf{N}$ such that, for every $f \in \mathcal{T}$, we have

$$\Theta(\widehat{\Phi}_j)(f, x) \leq \widehat{\Phi}_i(f, x).$$

Fix such x and f . By the definition of $\widehat{\Theta}$ and our assumption, we have

$$f(2^{\widehat{\Phi}_j(f, x)+1}) \leq \widehat{\Phi}_i(f, x).$$

Since there is no such query $f(2^{\widehat{\Phi}_j(f, x)+1}) = ?$ during the course of the computation of $\widehat{\Phi}_j(f, x)$, it follows that the value of f at $2^{\widehat{\Phi}_j(f, x)+1}$ has no effect on the value of $\widehat{\Phi}_j(f, x)$. Therefore, if $f(2^{\widehat{\Phi}_j(f, x)+1})$ is sufficiently large, then $\widehat{\Theta}(\widehat{\Phi}_j)(f, x) > \widehat{\Phi}_i(f, x)$. This contradicts our assumption. \square

Corollary 2. *There is a type-2 effective operator $\widehat{\Theta} : (\mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}) \rightarrow (\mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N})$ such that, for all computable $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$, we have*

$$\exists i : \widehat{\varphi}_i = F \forall j : \widehat{\varphi}_j = F \overset{\infty}{\exists}_2 (f, x) [\widehat{\Theta}(\widehat{\Phi}_j)(f, x) > \widehat{\Phi}_i(f, x)].$$

It is clear that Corollary 2 follows Theorem 6 immediately. Note that the corollary is the direct negation of the operator speed-up theorem.

6 Conclusion

In spite of the fact that oracle queries might interfere with the speed of an OTM, our investigation shows that the speed-up phenomena indeed exist in type-2 computation as long as the complexity measure satisfies Blum's two axioms. However, the phenomena disappear in higher-typed computation after type-2. We therefore have a strong belief that our investigation has completed the study of speed-up phenomena along the classical formulation of computational complexity, i.e., Blum's complexity measure. On the other hand, Blum's complexity measure may not be appropriate at type-2. For example, the query-complexity apparently fails to meet Blum's two axioms but it is such a commonly concerned resource in type-2 computation. Thus, a new approach is needed in understanding the concept of query-optimum programs. With a clear notion of query-optimum programs, we then can further examine the speed-up phenomena with respect to the notion of query-optimum programs. It would be interesting to continue research along this direction.

References

1. S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors. *Handbook of Logic in Computer Science*. Oxford University Press, 1992. Background: Mathematical Structures.
2. Manuel Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM*, 14(2):322–336, 1967.
3. Manuel Blum. On effective procedures for speeding up algorithms. *Journal of the ACM*, 18(2):290–305, 1971.
4. Critian Calude. *Theories of Computational Complexity*. Number 35 in Annals of Discrete Mathematics. North-Holland, Elsevier Science Publisher, B.V., 1988.
5. Stephen A. Cook. Computability and complexity of higher type functions. In *Logic from Computer Science*, pages 51–72. Springer-Verlag, 1991.
6. Nigel Cutland. *Computability: An introduction to recursive function theory*. Cambridge, New York, 1980.
7. Martin Davis, editor. *The Undecidable*. Raven Press, New York, 1965.
8. Kurt Gödel. Über die länge der beweise. *Ergebnisse eines Math. Kolloquiums*, 7:23–24, 1936. Translation in [7], pages 82–83, “On the length of proofs.”
9. J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transitions of the American Mathematics Society*, pages 285–306, May 1965.
10. Bruce M. Kapron and Stephen A. Cook. A new characterization of type 2 feasibility. *SIAM Journal on Computing*, 25:117–132, 1996.
11. Steve C. Kleene. Turing-machine computable functionals of finite types II. *Proceedings of London Mathematical Society*, 12:245–258, 1962.
12. Chung-Chih Li. Speed-up theorems in type-2 computation (full version). <http://www.itk.ilstu.edu/faculty/chungli/mypapers/SpeedUpFullVersion.pdf>.
13. Chung-Chih Li. Asymptotic behaviors of type-2 algorithms and induced baire topologies. In *Proceedings of the Third International Conference on Theoretical Computer Science*, pages 471–484, Toulouse, France, August 2004.
14. Chung-Chih Li. Clocking type-2 computation in the unit cost model. In *Proceedings of Computability in Europe: Logical Approach to Computational Barriers*, pages 182–192, Swansea, UK, 2006. Report# CSR 7-2006.
15. Chung-Chih Li and James S. Royer. On type-2 complexity classes: Preliminary report. pages 123–138, May 2001.
16. A. R. Meyer and P. C. Fischer. Computational speed-up by effective operators. *The Journal of Symbolic Logic*, 37:55–68, 1972.
17. Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967. First paperback edition published by MIT Press in 1987.
18. James S. Royer. Semantics vs. syntax vs. computations: Machine models of type-2 polynomial-time bounded functionals. *Journal of Computer and System Science*, 54:424–436, 1997.
19. Joel I. Seiferas. Machine-independent complexity theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 163–186. North-Holland, Elsevier Science Publisher, B.V., 1990. MIT press for paperback edition.
20. Anil Seth. Complexity theory of higher type functionals. Ph.d. dissertation, University of Bombay, 1994.
21. P. Van Emde Boas. Ten years of speed-up. *Proceedings of the Fourth Symposium Mathematical Foundations of Computer Science 1975*, pages 13–29, 1975. Lecture Notes in Computer Science.
22. Klaus Wagner and Gerd Wechsung. *Computational Complexity*. Mathematics and its applications. D. Reidel Publishing Company, Dordrecht, 1985.