Computability in Europe 2007

Computation and Logic in the Real World

University of Siena, June 18-23, 2007

Title:

Speed-up Theorems in Type-2 Computation

Abstract:

A classic result known as the speed-up theorem [2, 3] in the classical complexity theory shows that there exist some computable functions that do not have best programs for them. In this paper we lift this result into type-2 computation under the notion of our type-2 complexity theory depicted in [14, 12, 13]. 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 cancelation 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 [15] does not hold, which suggests that this curious phenomenon disappears in higher-typed computation beyond type-2.

Keywords:

Type-2 Complexity Theory, Type-2 Computation, Speed-up Theorems

Author:

Chung-Chih Li, Ph.D.
Assistant Professor
School of Information Technology

P.O. Box 5150
Old Union
Illinois State University
Normal, IL 61790, USA

Tel: +1-309-438-7952
Fax: +1-309-438-5113

e-mail: cli2@ilstu.edu

Speed-up Theorems in Type-2 Computation

Chung-Chih Li

School of Information Technology
Illinois State University
Normal, IL 61790, USA

Abstract. A classic result known as the speed-up theorem [2, 3] in the classical complexity theory shows that there exist some computable functions that do not have best programs for them. In this paper we lift this result into type-2 computation under the notion of our type-2 complexity theory depicted in [14, 12, 13]. 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 cancelation 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 [15] 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 in logical form was first remarked by Gödel [8].[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* [16] 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*

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

[1] The original remarks were translated and collected 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 [20].

[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 sketch our version of proof in Appendix A which will be used as a template proof for our type-2 speed-up theorem. More original proofs can be found in [2, 3, 20, 6, 21, 4, 18].

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 [15] 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 [15]). *For any total effective operator Θ, there is a recursive function f that can be uniformly constructed such that*

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

Our goal of the present paper is to lift these two speed-up theorems to type-2 computation. We obtain a type-2 analogy of Theorem 1. However, Theorem 2 fails to hold in the context of type-2 computation, which suggests that there always exist the 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 conventions and our setups. These paragraphs are perforce brief and superficial due to the space constraints. Also, as the matter of fact that the speed-up theorem is rather independent from the other part of the theory, our coverage will be very limited to related topics only. More details should be found in [14, 12, 13].

2 Conventions & Type-2 Complexity Theory

We consider natural numbers as type-0 objects and functions over natural numbers as type-1 objects. For type-2 objects, they are *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 $\mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$, where $\mathcal{T}$ is the set of total functions and $\rightharpoonup$ means possibly partial. Note that $f \in \mathcal{T}$ may not be computable. For $n \in \mathbf{N}$, $|n|$ 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 and then the oracle will return the answer to the question left on the query-tape and the answer will be prepared on the answer-tape for the OTM to read. All these 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 $\langle\widehat{\varphi}_i\rangle_{i\in\mathbf{N}}$ 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 $\widehat{M}_e$. Upon these agreements, in [19] Seth followed Hartmanis and Stearns's notion [9] to define type-2 complexity classes. He proposed two alternatives:

1. Given recursive $t : \mathbf{N} \to \mathbf{N}$, 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 $m = |\mathsf{max}(\{x\}\cup Q)|$ and Q is the set of all answers returned from the oracle during the course of the computation.
2. Given computable functional $H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, 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 [17]. 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 [12, 13] we adapted all these ideas and extended the second-ordered polynomial to a general type-2 computable functional to have the following complexity class:

$$DTIME(H) = \{F \mid \exists e[\widehat{\varphi}_e = F \text{ and } \widehat{\Phi}_e \leq^*_{\mathbf{2}} H]\}. \tag{1}$$

The notion of $\leq^*_{\mathbf{2}}$ used above is crucial to our works and will be formally defined later in this section. Along the line of the classical complexity theory initiated by a series of seminal papers [9, 2, 3], our previous results in [14, 12, 13] show that the complexity theory at type-2 is not parallel to its type-1 counterpart. To begin with, we defined $\leq^*_{\mathbf{2}}$ with a workable and reasonable type-2 analogy 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 [13] 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 $\mathsf{dom}(\sigma) \subset \mathbf{N}$ and $\mathsf{card}(\sigma) \in \mathbf{N}$. Given $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, let $F(f, x) \downarrow = y$ denote the case that F is defined at (f, x) and its value is y.

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

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

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 of 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 $(\sigma, x) \in \mathcal{T} \times \mathbf{N}$, 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 $k \in \mathsf{dom}(\sigma) \cap \mathsf{dom}(\tau)$ such that $\sigma(k) \neq \tau(k)$. In stead of taking every $(\!(\sigma, x)\!)$ with $\sigma \in \mathcal{F}$ as the basic open set[3], 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 $\mathbb{T} \times \mathbb{N}$ 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_3, 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. Now, we are in a position to define our type-2 almost-everywhere relation.

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

$$F_1 \leq^*_{\mathbf{2}} F_2 \quad \textit{if and only if} \quad X_{[F_1 \leq F_2]} \quad \textit{is co-compact in} \quad \mathbb{T}(F_1, F_2).$$

The complement of $X_{[F_1 \leq F_2]}$ is $X_{[F_1 > F_2]}$. The set $X_{[F_1 > F_2]}$ is called the *exceptional set* of $F_1 \leq^*_{\mathbf{2}} F_2$, i.e., $(f, a) \in X_{[F_1 > F_2]}$ iff $F_1(f, a) > F_2(f, a)$.

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

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 cancelation 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 cancelation 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 $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ defined by,

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

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, while has nothing to do with the final value, 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 $\leq_{\mathbf{2}}^{*}$ 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 $\Theta : \mathcal{T} \to \mathcal{T}$ by a functional $R : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$.

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

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

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 $r : \mathbf{N} \to \mathbf{N}$, there exists a computable functional $F_r : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that,*

$$\forall\, i : \widehat{\varphi}_i = F_r\ \exists j : \widehat{\varphi}_j = F_r\ [r \circ \widehat{\Phi}_j \leq_{\mathbf{2}}^{*} \widehat{\Phi}_i].$$

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

$$\forall\, i : \widehat{\varphi}_i = F_R\ \exists j : \widehat{\varphi}_j = F_R\ [\lambda f, x.R(f, x, \widehat{\Phi}_j(f, x)) \leq_{\mathbf{2}}^{*} \widehat{\Phi}_i].$$

Theorem 3 and Theorem 4 are lifted from Theorem 1 and Theorem 2, respectively. 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, $r \circ \widehat{\Phi}_j \leq_{\mathbf{2}}^{*} \widehat{\Phi}_i$, 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 $\leq_{\mathbf{2}}^{*}$ in the theorem is the coarsest one, i.e., the topology with basic open sets: $(\!(\emptyset, 0)\!), (\!(\emptyset, 1)\!), \ldots$. 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 cancelation 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 $r \circ \widehat{\Phi}_j \leq^*_{\mathbf{2}} \widehat{\Phi}_i$ 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 in 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 $r : \mathbf{N} \to \mathbf{N}$, there exists a computable functional $F_r : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that, for every $\widehat{\varphi}$-program i for F_r, there is another $\widehat{\varphi}$-program j such that,*

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

Due to the space constraints, detailed arguments for this pseudo-speed-up theorem are presented in Appendix B.

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 $\widehat{\varphi}_i =^*_{\mathbf{2}} \widehat{\varphi}_j$ and $r \circ \widehat{\Phi}_j \leq^*_{\mathbf{2}} \widehat{\Phi}_i$ does not hold in general because $((\emptyset, x))$ may not be a basic open set for some x. Consider the following exception set

$$E = \left\{(f,x) \;\middle|\; \widehat{\varphi}_i(f,x) \neq \widehat{\varphi}_j(f,x)\right\}.$$

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 loop-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, the exception set

$$E' = \left\{(f,x) \;\middle|\; r \circ \widehat{\Phi}_j(f,x) > \widehat{\Phi}_i(f,x)\right\}$$

has $\{x|(f,x) \in E'\}$ finite. Consider the above patched $\widehat{\varphi}$-program, j'. The exception set

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

must have $\{x|(f,x) \in E''\}$ finite. Therefore, E'' is compact in the relative topology $\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_{j'})$, 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. □

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 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 to such theorem needs a general type-2 s-m-n and a type-2 recursion theorem, which we don't have. We shall prove 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 $(\mathcal{T} \times \mathbf{N} \to \mathbf{N}) \to (\mathcal{T} \times \mathbf{N} \to \mathbf{N})$ with inputs restricted to computable total type-2 functionals. Thus, we can think up that an effective type-2 operator is given 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.

Recall Definition 3 for $\leq_{\mathbf{2}}^*$, the following type-2 quantifiers use the same idea of compactness. For continuous functionals $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, we have $\overset{\infty}{\forall}_{\mathbf{2}}\ (f,x)[F(f,x) \leq G(f,x)]$ 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 $\overset{\infty}{\exists}_{\mathbf{2}}\ (f,x)[F(f,x) \leq G(f,x)]$ 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

$$\begin{aligned} F \leq_{\mathbf{2}}^* G &\iff \overset{\infty}{\forall}_{\mathbf{2}}\ (f,x)[F(f,x) \leq G(f,x)] \\ &\iff \neg\ \overset{\infty}{\exists}_{\mathbf{2}}\ (f,x)[F(f,x) > G(f,x)]. \end{aligned}$$

When the concerned functionals F and G are clear from the context, we simply read $\overset{\infty}{\forall}_{\mathbf{2}}\ (f,x)$ as *"for all (f,x) except those in a compact set such that"*, and $\overset{\infty}{\exists}_{\mathbf{2}}\ (f,x)$ 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.*

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

$$\forall i : \widehat{\varphi}_i = F \; \forall j : \widehat{\varphi}_j = F \;\; \overset{\infty}{\exists}_{\mathbf{2}} \, (f, x)[\widehat{\Theta}(\widehat{\Phi}_j)(f, x) > \widehat{\Phi}_i(f, x)].$$

Proof: Define $\widehat{\Theta} : (\mathcal{T} \times \mathbf{N} \to \mathbf{N}) \to (\mathcal{T} \times \mathbf{N} \to \mathbf{N})$ 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 $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$. 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., $\widehat{\Theta}(\widehat{\Phi}_j) \leq^*_{\mathbf{2}} \widehat{\Phi}_i$. 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. □

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

$$\exists i : \widehat{\varphi}_i = F \; \forall j : \widehat{\varphi}_j = F \;\; \overset{\infty}{\exists}_{\mathbf{2}} \, (f, x)[\widehat{\Theta}(\widehat{\Phi}_j)(f, x) > \widehat{\Phi}_i(f, x)].$$

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 the speed of an OTM is interfered by the oracle queries, 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 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. Nevertheless, since the oracle-query is such a special and unique complexity measure that is seen only at type-2, we speculate that such complexity can provide a new prospect to look at the speed-up phenomena at type-2. Apparently, query-complexity fails to meet Blum's two axioms, and hence a new approach is needed in understanding the concept of query-optimum programs. It would be interesting to take up the speculation in 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. 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.
13. 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.
14. Chung-Chih Li and James S. Royer. On type-2 complexity classes: Preliminary report. pages 123–138, May 2001.
15. A. R. Meyer and P. C. Fischer. Computational speed-up by effective operators. *The Journal of Symbolic Logic*, 37:55–68, 1972.
16. Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967. First paperback edition published by MIT Press in 1987.
17. 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.
18. 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.
19. Anil Seth. Complexity theory of higher type functionals. Ph.d. dissertation, University of Bombay, 1994.
20. 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.
21. Klaus Wagner and Gerd Wechsung. *Computational Complexity*. Mathematics and its applications. D. Reidel Publishing Company, Dordrecht, 1985.

Appendix

A Proof of The Speed-up Theorem

We sketch the original proof of the Speed-up Theorem that has been customized so that we can easily lift it into a type-2 analog. A full version of the proof can be found in any of [2, 3, 20, 6, 21, 4, 18]. We start with an intermediate theorem known as "Pseudo-Speed-up Theorem."

Theorem 7 (Pseudo-Speed-up Theorem). *Let $r : \mathbf{N} \to \mathbf{N}$ be recursive. There exists a recursive function $f_r : \mathbf{N} \to \mathbf{N}$ such that,*

$$\forall\, i : \varphi_i = f_r \exists j : \varphi_j =^* f_r \ \overset{\infty}{\forall}\ x[\Phi_i(x) > r \circ \Phi_j(x)].$$

Fix any recursive function $r : \mathbf{N} \to \mathbf{N}$. Let s be an s-1-1 function such that, for all $e, u, x \in \mathbf{N}$, $\varphi_{s(e,u)}(x) = \varphi_e(u, x)$. We shall construct, by the recursion theorem, a φ-program e such that,

a) $\varphi_e : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$,
b) for every $u \in \mathbf{N}$, for all but finitely many $x \in \mathbf{N}$, $\varphi_e(0, x) = \varphi_e(u, x)$, and
c) for every $i \in \mathbf{N}$, if $\varphi_i = \varphi_{s(e,0)}$, then $\varphi_i =^* \varphi_{s(e,i+1)}$ and $r \circ \Phi_{s(e,i+1)} <^* \Phi_i$.

Given such a φ-program e, the speedable recursive function f_r is the function computed by the φ-program $s(e, 0)$, i.e., $\lambda x.\varphi_e(0, x)$, and, for each φ-program i for f_r, $s(e, i + 1)$ is a sped-up finite variant of the φ-program i. The theorem is called the "Pseudo" Speed-up Theorem because $s(e, i + 1)$ is not an exact sped-up version of f_r but just computes f_r almost everywhere.

We maintain a global cancelation set $C_{u,x}$ for each $u, x \in \mathbf{N}$. The cancelation set, $C_{u,x}$, determines the value of $\varphi_e(u, x)$. $C_{u,x}$ is defined recursively based on:

1. The previously defined sets: $C_{u,u},\ C_{u,u+1},\ \ldots,\ C_{u,x-1}$, and
2. the cost of computing each of $\varphi_{s(e,u+1)}(x),\ \varphi_{s(e,u+2)}(x),\ \ldots,\ \varphi_{s(e,x)}(x)$.

Figure 1 shows the dependence of $C_{u,x}$ on these prior sets and run times. Precisely, for each $u, x \in \mathbf{N}$, $\varphi_e(u, x)$ and $C_{u,x}$ are defined as follows.

a) If $x \leq u$, then set $C_{u,x} = \emptyset$ and $\varphi_e(u, x) = 1$.
b) If $x > u$, then set $\varphi_e(u, x) = 1 + \mathsf{max}(\{\varphi_i(x) \mid i \in C_{u,x}\})$, where

$$C_{u,x} = \left\{ i \,\middle|\, \begin{array}{l} u \leq i < x \text{ and } i \notin C_{u,u} \cup C_{u,u+1} \cup \cdots \cup C_{u,x-1} \\ \text{and } \Phi_i(x) \leq r \circ \Phi_{s(e,i+1)}(x) \end{array} \right\}.$$

Claims:

1. φ_e is total.
2. For every $u, x \in \mathbf{N}$, $C_{u,x} = C_{0,x} \cap \{u, u + 1, \ldots, x - 1\}$.
3. For every $u, x_1, x_2 \in \mathbf{N}$, if $x_1 \neq x_2$, then $C_{u,x_1} \neq C_{u,x_2}$.
4. For every $u \in \mathbf{N}$, for all but finitely many $x \in \mathbf{N}$, $\varphi_e(0, x) = \varphi_e(u, x)$.

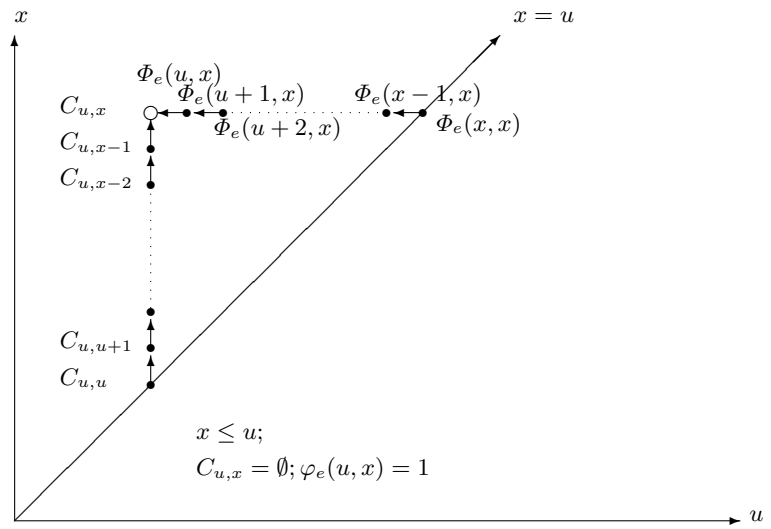

Fig. 1. Construction of $C_{u,x}$

5. For every $i \in \mathbf{N}$, if φ_i computes $\varphi_{s(e,0)}$, then $\varphi_i =^* \varphi_{s(e,i+1)}$ and there exists $n_0 \in \mathbf{N}$ such that, for every $x \geq n_0$, we have $\Phi_i(x) > r \circ \Phi_{s(e,i+1)}(x)$.

Proofs of the Claims:

1. For $x \leq u$, $\varphi_e(u, x)$ and $C_{u,x}$ are defined to be 1 and $\emptyset$, respective. For $x > u$, Figure 1 shows that every such point is well defined based on some finite previously defined points and cancelation sets.

2. We prove this claim by double induction on u and x as follows.

   Basis: Clearly, if $x = 0$, then for every $u \in \mathbf{N}$, $C_{u,0} = C_{0,0} \cap \emptyset = \emptyset$.

   Hypothesis: Fix any $n \in \mathbf{N}$. Assume that if $x \leq n$, then for every $u \in \mathbf{N}$, $C_{u,x} = C_{0,x} \cap \{u, u+1, \ldots, x-1\}$.

   Inductive step: We argue that, when $x = n+1$, then $C_{u,x} = C_{0,x} \cap \{u, u+1, \ldots, x-1\}$ for each $u \in \mathbf{N}$. Without loss of generality, we can assume that $u < n+1$, for the $u \geq n+1$ case is trivial. Thus, we argue that,

$$\forall u < n+1 [C_{u,n+1} = C_{0,n+1} \cap \{u, u+1, \ldots, n\}].$$

Given $i \in C_{u,n+1}$, we have:

$$\begin{aligned}
i \in C_{u,n+1} &\iff i \in \{u, u+1, \ldots, n\}, \\
&\quad\ i \notin C_{u,u} \cup C_{u,u+1} \cup \cdots \cup C_{u,n}, \text{ and} \\
&\quad\ \Phi_i(n+1) \leq r \circ \Phi_{s(e,i+1)}(n+1) \\
&\iff i \in \{u, u+1, \ldots, n\}, \\
&\quad\ i \in \{0, 1, \ldots, u, u+1, \ldots, n\}, \\
&\quad\ i \notin C_{u,u} \cup C_{u,u+1} \cup \cdots \cup C_{u,n}, \text{ and} \\
&\quad\ \Phi_i(n+1) \leq r \circ \Phi_{s(e,i+1)}(n+1) \\
&\iff i \in \{u, u+1, \ldots, n\}, \\
&\quad\ i \in \{0, 1, \ldots, u, u+1, \ldots, n\}, \\
&\text{by hypothesis} \quad i \notin (C_{0,u} \cup C_{0,u+1} \cup \cdots \cup C_{0,n}) \cap \{u, u+1, \ldots, n-1\}, \\
&\quad\ \text{and } \Phi_i(n+1) \leq r \circ \Phi_{s(e,i+1)}(n+1) \\
&\iff i \in \{u, u+1, \ldots, n\} \text{ and } i \in C_{0,n+1} \\
&\iff i \in C_{0,n+1} \cap \{u, u+1, \ldots, n\}.
\end{aligned}$$

3. Let $x_1 \neq x_2$. From the construction of the cancelation sets, it is clear that $i \in C_{u,x_1} \Rightarrow i \notin C_{u,x_2}$ and $i \in C_{u,x_2} \Rightarrow i \notin C_{u,x_1}$.
4. For every $u, x \in \mathbf{N}$, the values of $\varphi_e(0,x)$ and $\varphi_e(u,x)$ are determined by $C_{0,x}$ and $C_{u,x}$, respectively. By Claim **2**, $C_{0,x} - C_{u,x} \subseteq \{0, 1, \ldots, u-1\}$. Thus, only indices in $\{0, 1, \ldots, u-1\}$ may cause the difference between $C_{0,x}$ and $C_{u,x}$. But, by Claim **3**, each such index will be selected at most once for some x. Thus, if x is sufficiently large, all indices in $\{0, 1, \ldots, u-1\}$ will have been canceled and will not be selected again, and hence $C_{0,x} = C_{u,x}$.
5. Suppose that $\varphi_i = \varphi_{s(e,0)}$. From Claim **4**, we already have $\varphi_i =^* \varphi_{s(e,i+1)}$. For the other part of this claim, we assume, by contradiction, there are infinitely many x such that, $\Phi_i(x) \leq r \circ \Phi_{s(e,i+1)}(x)$. Then for some sufficiently large a with $a \geq i$, i will be selected into the cancelation set $C_{0,a}$. Hence, $\varphi_i(a) \neq \varphi_{s(e,0)}(a)$. This is a contradiction.

This completes the proof the Pseudo-Speed-up Theorem. □

To obtain the Speed-up Theorem, we can patch the almost everywhere equality in the Pseudo-Speed-up Theorem by means of a finite table that stores the exact values of the speedable function on those exceptional points. However, the finite table cannot be uniformly constructed, and hence the proof of the Speed-up Theorem is not constructive in this sense.[4]

B Proof of Type-2 Pseudo-Speed-up Theorem

Theorem (Type-2 Pseudo-Speed-up Theorem) *For any recursive function function $r : \mathbf{N} \to \mathbf{N}$, there exists a computable functional $F_r : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such*

[4] A rather comprehensive discussion about the constructibility of the proof of the Speed-up Theorem can be found in [20].

that, for every $\widehat{\varphi}$-program i for F_r, there is another $\widehat{\varphi}$-program j such that,

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

Fix a recursive function $r : \mathbf{N} \to \mathbf{N}$. With the s-m-n and recursion theorems on the type-0 argument introduced, let s be an s-1-2 function such that, for every $e, u, x \in \mathbf{N}$ and $f \in \mathcal{T}$, $\widehat{\varphi}_{s(e,u)}(f,x) = \widehat{\varphi}_e(u,f,x)$. We shall construct, by the recursion theorem, a $\widehat{\varphi}$-program e that is similar to the φ-program in Theorem 7, such that:

a) $\widehat{\varphi}_e : \mathbf{N} \times \mathcal{T} \times \mathbf{N} \to \mathbf{N}$.
b) For every $u \in \mathbf{N}$, there exists $n_0 \in \mathbf{N}$ such that, for every $x > n_0$ and $f \in \mathcal{T}$, we have $\widehat{\varphi}_e(0,f,x) = \widehat{\varphi}_e(u,f,x)$.
c) For every $i \in \mathbf{N}$, if $\widehat{\varphi}_i$ computes $\widehat{\varphi}_{s(e,0)}$, then there exists $n_0 \in \mathbf{N}$ such that, if $x \in \mathbf{N}$ with $x > n_0$, then for every $f \in \mathcal{T}$, $\widehat{\varphi}_i(f,x) = \widehat{\varphi}_{s(e,i+1)}(f,x)$ and $\widehat{\Phi}_i(f,x) > r \circ \widehat{\Phi}_{s(e,i+1)}(f,x)$.

Clearly, such $\widehat{\varphi}_e$ witnesses our Type-2 Pseudo-Speed-up Theorem. Similarly, we maintain a global cancelation set $C_{u,x}$ for each $u, x \in \mathbf{N}$, which is defined as follows. Let $f_0 = \lambda x.0$. Suppose that $u, x \in \mathbf{N}$ and $f \in \mathcal{T}$.

a) If $x \leq u$, then let $C_{u,x} = \emptyset$ and $\widehat{\varphi}_e(u,f,x) = 1$.
b) If $x > u$, then define $C_{u,x}$ by:

$$C_{u,x} = \left\{ i \;\middle|\; \begin{array}{l} u \leq i < x \text{ and } i \notin C_{u,u} \cup C_{u,u+1} \cup \cdots \cup C_{u,x-1} \text{ and} \\ \left[\begin{array}{l} \widehat{\Phi}_i(f_0,x) \leq r \circ \widehat{\Phi}_{s(e,i+1)}(f_0,x) \ \text{ or } \ \text{the OTM}, \widehat{M}_i, \\ \text{on } (f_0,x), \text{ makes at least one query in } i \text{ steps} \end{array} \right] \end{array} \right\},$$

and define $\widehat{\varphi}_e(u,f,x)$ by:

$$\widehat{\varphi}_e(u,f,x) = 1 + \mathsf{max}(\{\widehat{\varphi}_i(f_0,x) \mid i \in C_{u,x}\}). \tag{3}$$

In addition to the five claims in the proof of Theorem 7, we have one more claims to our construction. Consider the following six claims.

1. $\widehat{\varphi}_e$ is total on $\mathbf{N} \times \mathcal{T} \times \mathbf{N}$.
2. For every $u, x \in \mathbf{N}$, $C_{u,x} = C_{0,x} \cap \{u, u+1, \ldots, x-1\}$.
3. For every $u, x_1, x_2 \in \mathbf{N}$, if $x_1 \neq x_2$, then $C_{u,x_1} \neq C_{u,x_2}$.
4. For every $u \in \mathbf{N}$, for all but finitely many $x \in \mathbf{N}$, and for every $f \in \mathcal{T}$, $\widehat{\varphi}_e(0,f,x) = \widehat{\varphi}_e(u,f,x)$.
5. For every $i \in \mathbf{N}$, if $\widehat{\varphi}_i = \widehat{\varphi}_{s(e,0)}$, then there exists $n_0 \in \mathbf{N}$ such that, for every $x \in \mathbf{N}$ with $x \geq n_0$ and for every $f \in \mathcal{T}$, we have $\widehat{\varphi}_i(f,x) = \widehat{\varphi}_{s(e,i+1)}(f,x)$ and $\widehat{\Phi}_i(f,x) > r \circ \widehat{\Phi}_{s(e,i+1)}(f,x)$.
6. If i is a $\widehat{\varphi}$-program for $\widehat{\varphi}_{s(e,0)}$, then, for all but finitely many $x \in \mathbf{N}$ and for all $f \in \mathcal{T}$, the OTM $\widehat{M}_i$, on (f,x), does not make any oracle query.

For claim 1, it is clear that the extra clause

$$\text{“the OTM, } \widehat{M}_i, \text{ on } (f_0, x), \text{ makes at least one query in } i \text{ steps”}$$

in defining $C_{u,x}$ is recursively decidable, and hence $\widehat{\varphi}_e$ is total.

Claims 2, 3, 4, and 5 can be proved by exactly the same arguments for Theorem 7.

For Claim 6, suppose $\widehat{\varphi}_i = \widehat{\varphi}_{s(e,0)}$ and, by contradiction, there are infinitely many $x \in \mathbf{N}$ such that, for some $f \in \mathcal{T}$, the OTM, $\widehat{M}_i$, on (f, x), makes some queries to the oracle. Let a be such x. Then, $\widehat{M}_i$, on (f_0, a), must also make some queries to the oracle. Moreover, there are infinitely many $\widehat{\varphi}$-programs that behavior exactly the same as i does. Let j be a such $\widehat{\varphi}$-program and sufficiently large. Thus, $\widehat{M}_j$, on (f_0, a), will make some queries in j steps and will be selected into $C_{0,a}$. Therefore, j and i are not $\widehat{\varphi}$-program for $\widehat{\varphi}_{s(e,0)}$. □