

Speed-up Theorems in Type-2 Computations Using Oracle Turing Machines

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 computations. Although the speed-up phenomenon is essentially inherited from type-1 computations, we observe that a direct application of the original proof to our type-2 speed-up theorem is problematic because the oracle queries can interfere with 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 [18] does not hold, which suggests that this curious speed-up phenomenon disappears in higher-typed computations beyond type-2. The result of this paper adds one more piece of evidence to support the general type-2 complexity theory under the framework proposed in [15–17] as a reasonable setup.

1 Introduction

Speed-up phenomena have been extensively studied by mathematicians for more than a half century, beginning with Gödel’s work on the length of proofs [9].¹ Gödel pointed out that by adding some additional axiom to a system S_i , we not only obtain a bigger system S_{i+1} (i.e., there exist theorems that are provable in S_{i+1} using the new axiom but not provable in S_i), we also arbitrarily shorten (speed-up) the length of infinitely many proofs in S_i . In [2], 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 other words, there is a function, such that whenever we construct a program for it, there always exist “better” programs that run arbitrarily faster than the one we just constructed. The relation between Gödel’s observation and Blum’s discovery was not obvious at all, but it is clear that Blum’s work alone has played an important role in machine independent complexity theory for the past few decades.²

¹ The original remarks were translated in [7], pages 82-83.

² More discussion about the relation between computational speed-up phenomena and Gödel’s speed-up results in logic can be found in [29].

In order to state the theorem precisely, we first fix some notation and convention. Let \mathbf{N} be the set of natural numbers. 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 with the e^{th} Turing machine. In his seminal papers [2, 3], Blum postulated two intuitive requirements for computational complexity measure in the following axioms:

Axiom 1: $\forall e, x \in \mathbf{N} [\varphi_e(x) \downarrow \iff \Phi_e(x) \downarrow]$.

Axiom 2: $\forall e, x, m \in \mathbf{N} \Phi_e(x) \leq m$ is decidable.

The first axiom requires that, for any φ -program e on any input x , if the program converges (halts), then the cost of computing $\varphi_e(x)$ also converges. The second axiom requires that, if a finite upper bound on the resource is fixed, we then can effectively decide if a given computation will halt within the resource bound. Almost every reasonable complexity measure for type-1 computations satisfies the two axioms. For example, time complexity clearly satisfies the two axioms. Also, we can manage space complexity to satisfy the axioms as well. If a proposed resource measure satisfies Blum's two axioms, we say that it is a *complexity measure*. More precisely, let $\langle \varphi_i \rangle_{i \in \mathbf{N}}$ be an *acceptable programming system* [22] and $\langle \Phi_i \rangle_{i \in \mathbf{N}}$ be a *complexity measure* associated with $\langle \varphi_i \rangle_{i \in \mathbf{N}}$. 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*

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

The standard asymptotic notion, $\overset{\infty}{\forall}$, is read as *for all but finitely many*.³ We say that the recursive function r in the theorem is a speed-up factor, and the function f is *r-speedupable*. The original proof of this theorem is given in [2, 3], and some nice revisions can be found in [4, 6, 25, 29, 30]. Many variations of the speed-up theorem have since been proven. We are interested in Meyer and Fischer's operator speed-up theorem [18] where the speed-up factor is strengthened by an effective operator Θ as follows:

Theorem 2 (The Operator Speed-up Theorem [18]). *For any total effective operator Θ , there is a recursive function f such that*

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

³ Precisely, $\overset{\infty}{\forall} x P(x)$ is equivalent to $\exists x_0 \forall x (x_0 \leq x \rightarrow P(x))$. The negation of “for all but finitely many” is read as “exist infinitely many” denoted by $\overset{\infty}{\exists}$. In other words, $\overset{\infty}{\exists} x P(x)$ is equivalent to $\forall x_0 \exists x (x_0 \leq x \wedge \neg P(x))$.

This Operator Speed-up Theorem is a stronger theorem in two aspects. First, the proof of the operator speed-up theorem is constructive, i.e., the speedupable function can be uniformly constructed while Blum’s speedupable function cannot. Secondly, the operator Θ takes the graph of the computation of $\Phi_j(x)$ but the speed of computing $\varphi_j(x)$ is still out of its control and hence the speedup phenomenon remains.

Our goal in the present paper is to lift these two speed-up theorems into type-2 computations. We obtain a type-2 analog of Theorem 1. However, Theorem 2 fails to hold in the context of type-2 computations, which suggests that there always exist best programs in higher-typed computations 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 performed brief and superficial due to space constraints. Also, 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 [15–17].

2 Type-2 Complexity Theory & Conventions

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 $\mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$, 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 $n \in \mathbf{N}$, $|n|$ denotes the length of the binary bit string representing n .

For type-2 computations, 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 single-taped TM, 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 the query-state, in which the oracle will read the query that is left on the query-tape and prepare its answer on the answer-tape for the OTM to read. Since how the oracle fetches the question and provides the answer to it is not the OTM’s concern, we charge one unit cost (i.e., one step) to the OTM for this process. Note that the oracle may not be computable. However, the OTM has to prepare the queries and read their answers at its own cost. Thus, if a resource bound is provided, the OTM can’t make an arbitrarily large oracle query, since writing the query to the query-tape will use up its resource. We also fix a programming system $\langle \hat{\varphi}_i \rangle_{i \in \mathbf{N}}$ associated with some complexity measure $\langle \hat{\Phi}_i \rangle_{i \in \mathbf{N}}$ 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 \tilde{M}_e to denote the OTM with index e and $\hat{\varphi}_e$ to denote

the functional computed by \widehat{M}_e . Following these conventions, Seth [26] adapted Hartmanis and Stearns's notion [10] and defined 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 [10]. The same machine characterization idea can also be found in other works such as Kapron and Cook's [11] and Royer's [24]. In Seth's first definition stated above, the resource bound is determined by the maximum size of all oracle answers returned during the computation. But the difficulty is that 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 proposed their remedies in the context of feasible functionals and gave a very neat characterization for the type-2 Basic Feasible Functionals (BFF) in [11], where the so-called second-ordered polynomials are used as the resource bounds. In [15, 16] we adapted all these ideas and extended the second-ordered polynomials to general type-2 computable functionals. We proposed the following complexity class: For any computable type-2 functional H ,

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

The relation, \leq_2^* , used above will be defined in Definition 3, which is crucial to our work. Along the lines of classical complexity theory initiated by a series of seminal papers [10, 2, 3], our previous results in [15–17] show that the complexity theory at type-2 does not parallel its type-1 counterpart. To begin with, we defined \leq_2^* with a workable and reasonable type-2 analog of type-1 asymptotic notion. We equated our notion of *finitely many* at type-2 to the *compact sets* in some Baire-like topology [1]. The difficulty of using the standard Baire topology is that the Baire topology is too fine and hence the nonempty compact sets in it are not computable. We thus reduced the Baire topology into a relative one that is defined depending on the functionals concerned. This seemingly less universal topology turns out to be the only workable framework for type-2 complexity.

As there is no type-2 equivalent of the Church-Turing thesis, the compactness in our definition is the key property to guarantee that our construction is

computable. In [16] 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 such as the Gap Theorem and the Union Theorem, the speed-up theorems do not need a precise definition of complexity classes. We thus skip the detailed definition of our explicit type-2 complexity classes in this paper. However, the asymptotic notion is still indispensable in the speed-up theorems. We formalize the notion in the following paragraphs.

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 . Two important properties of computable functionals are *compactness* and *monotonicity* defined as follows. For any $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$, we say that F is

- *compact* iff $\forall (f, x) \in (\mathbf{N} \rightarrow \mathbf{N}) \times \mathbf{N} \exists \sigma \in \mathcal{F} [F(f, x) = F(\sigma, x)]$;
- *monotone* iff $\forall (\sigma, x) \in \mathcal{F} \times \mathbf{N} [F(\sigma, x) \downarrow \Rightarrow \forall \tau \supseteq \sigma (F(\tau, x) \downarrow = F(\sigma, x))]$.

Using the theorem due to Uspenskii and Nerode [28, 19], we know that a functional F is continuous if and only if F is compact and monotone. Note that a continuous function may not be computable, but a computable function must be continuous due to the finite computation requirement. This can be described in terms of *locking fragments* defined as follows.

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 $\sigma(k) \neq \tau(k)$. Instead of taking every $((\sigma, x))$ with $\sigma \in \mathcal{F}$ as the basic open set⁴, we consider only those that are related to the concerned functionals as follows.

⁴ 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} .

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 $(f, a) \in \mathcal{T} \times \mathbb{N}$, (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 $X_{[F_1 \leq F_2]} \subseteq \mathcal{T} \times \mathbb{N}$ 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 $F_1, F_2 : \mathcal{T} \times \mathbb{N} \rightarrow \mathbb{N}$ be continuous. Define

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

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 $F, G : \mathcal{T} \times \mathbb{N} \rightarrow \mathbb{N}$, 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 every (f, x) in the set ...”, where *compact* is understood as $\mathbb{T}(F, G)$ -compact. Unfortunately, \forall_2^∞ and \exists_2^∞ can’t be defined in a general form because there is no workable general topology to define compactness for type-2 complexity. A detailed discussion regarding this concern can be found in [15].

3 Lifting Speed-up Theorems to Type-2

Since type-1 computations are just a special case of type-2 computations, the speedupable 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 $\langle \hat{\Phi}_i \rangle$, the number of steps the OTM performs, does satisfy Blum’s two axioms. However, we observe that oracle queries in type-2 computations have introduced some difficulties when we attempt a direct translation of the original proof. Recall that the original construction of the

speedupable 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 cancelled. Note that the proofs of the Union Theorem and Gap Theorem do not rely on 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 that do not affect the concerned topology but do affect the running time of the computation. 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 in the sense that whatever answer the oracle returns, the result of the computation will not be altered. 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 as follows: Fix a computable f . For any $a \in \mathbf{N}$, during the computation of F on (f, a) , if the OTM enters the query-state, then it outputs 1 and stops; if the OTM never enters the query-state, then it will eventually stop and output 0. Thus, if our procedure above outputs 1, that means $\varphi_a(a) \downarrow$; otherwise $\varphi_a(a) \uparrow$, which is impossible.

However, the answer to the query at $f(0)$ does affect the time for the machine to halt. The smaller $f(0)$ is, the sooner the computation halts. In fact, it is easy to construct a computable functional that makes unnecessary queries on all inputs, and moreover, the number of unnecessary queries can be arbitrarily large. Such unnecessary but speed-affecting queries are the problem we must get around in lifting the speed-up theorems into type-2.

It is clear that our $\hat{\varphi}$ -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 $\hat{\varphi}$ -programming system. To that speedupable function, any queries made during the course of the computation are unnecessary. However, as we have seen, unnecessary queries may affect the computation time. Therefore, we cannot simply cancel those $\hat{\varphi}$ -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 speedupable 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. We will see that our notion of \leq_2^* defined by Definition 3 based on the compactness of the relative topologies (Definition 2) can resolve this problem.

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 can be considered as 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,*

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

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

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,*

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

Theorem 3 is obtained by lifting Theorem 1 into type-2 computations. We observe that $F_r = \hat{\varphi}_i = \hat{\varphi}_j$. By Definition 3, the relative topology for the type-2 relation, $r \circ \hat{\Phi}_j \leq_2^* \hat{\Phi}_i$, is

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

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 $\mathbb{T}(F_r)$, then the type-0 component of the elements in S must have infinitely many different values. If a $\hat{\varphi}$ -program i needs to be canceled, we will 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, which is the required lemma in proving the speed-up theorem. Because, for every $\hat{\varphi}$ -program i for F_r , its *pseudo* speed-up version, $\hat{\varphi}$ -program j , does not exactly compute $\hat{\varphi}_i$ on some finitely many type-0 inputs, and hence $\hat{\varphi}_i$ and $\hat{\varphi}_j$ may define two different topologies. Thus, if we ignore the effect of the type-1 argument, the almost everywhere relation $r \circ \hat{\Phi}_j \leq_2^* \hat{\Phi}_i$ may fail in topology $\mathbb{T}(\hat{\varphi}_i, \hat{\varphi}_j)$. To fix this problem, we introduce a weaker type-2 pseudo-speed-up theorem, in which 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 4 (Type-2 Pseudo-Speed-up Theorem). *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, for every $\widehat{\varphi}$ -program i for F_r , there is another $\widehat{\varphi}$ -program j such that,*

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

To keep our discussion focused, we shall put detailed proof of this pseudo-speed-up theorem in the appendix. With this type-2 pseudo-speed-up theorem, we are able to prove Theorem 3.

Proof of Theorem 3: Let $f_0 = \lambda x.0$. According to the construction of $\widehat{\varphi}_e$ in Theorem 4, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{\varphi}_e(0, f, x) = \widehat{\varphi}_e(0, f_0, x)$. 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 $\widehat{\varphi}_i \stackrel{*}{=} \widehat{\varphi}_j$ and $r \circ \widehat{\Phi}_j \leq \widehat{\Phi}_i$ 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 $\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_j)$, $\{x \mid (f, x) \in E\}$ must be finite. Thus, we can construct a patched $\widehat{\varphi}$ -program j' such that the program will search a look-up table if the type-0 argument is in $\{x \mid (f, x) \in E\}$. 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 $x \notin \{x \mid (f, x) \in E\}$, then $\widehat{\varphi}$ -program j' starts running $\widehat{\varphi}$ -program j . Similarly, consider

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

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

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

so that $\{x \mid (f, x) \in E''\}$ 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 4, this problem can be ignored, because any program that will make any query on some inputs does not compute our speedupable functional. This completes the proof of Theorem 3. \square

5 Type-2 Operator Anti-speed-up Theorem

In the previous section we have proven a type-2 speed-up theorem. The speed-up factor in Theorem 3 is a type-1 function and the proof is adapted from the proof

for the original speed-up theorem. In this section we consider a type-2 analog of the operator speed-up theorem by lifting the operator in Theorem 2 into a higher typed operator that takes functionals as its input. In other words, we will try to explore a speed-up phenomenon when the speed-up factor is type-3. Clearly, a proof of such a theorem needs a general type-2 s-m-n theorem and a general type-2 recursion theorem, but such two theorems require a type-2 Church-Turing thesis, which we don't have. Instead, we shall 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 [13] 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 of an effective type-2 operator as 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 $\widehat{\Theta}$ -speed-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 5 (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*

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

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 $\widehat{\Theta}$ -speed-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

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

Clearly, there must be no query to f -oracle at $2^{\widehat{\Phi}_j(f, x)+1}$ during the course of the computation of $\widehat{\Phi}_j(f, x)$, because otherwise the cost of making such query will be higher than $\widehat{\Phi}_j(f, x)$, which is impossible. Thus, 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 every computable $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$, we have*

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

Since $\forall x P(x) \rightarrow \exists x P(x)$, it is clear that Corollary 2 follows Theorem 5 immediately. The corollary is of no interest but shows the direct negation of the operator speed-up theorem.

6 Conclusions

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 computations as long as the complexity measure satisfies Blum's two axioms. Thus, one major significance of this paper should be seen as a piece of evidence to support the general type-2 complexity theory under the framework proposed in [15–17], which we consider a reasonable setup in a sense that both the familiar complexity structure and proof techniques used in classical complexity theory are mostly preserved. On the other hand, the phenomena disappear in higher-typed computations 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. However, 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 a commonly concerned resource in type-2 computations. 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. Cristian 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. R.O. Gandy and J.M.E. Hyland. Computable and recursively countable functions of higher type. *Logic Colloquium*, 76:405–438, 1977.
9. Kurt Gödel. Über die länge der beweis. *Ergebnisse eines Math. Kolloquiums*, 7:23–24, 1936. Translation in [7], pages 82-83, “On the length of proofs.”.
10. J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transitions of the American Mathematics Society*, pages 285–306, May 1965.
11. Bruce M. Kapron and Stephen A. Cook. A new characterization of type 2 feasibility. *SIAM Journal on Computing*, 25:117–132, 1996.
12. Steve C. Kleene. Recursive functionals and quantifiers of finite types I. *Transitions of the American Mathematics Society*, 91:1–52, 1959.
13. Steve C. Kleene. Turing-machine computable functionals of finite types II. *Proceedings of London Mathematical Society*, 12:245–258, 1962.
14. Steve C. Kleene. Recursive functionals and quantifiers of finite types II. *Transitions of the American Mathematics Society*, 108:106–142, 1963.
15. 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.
16. 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.
17. Chung-Chih Li and James S. Royer. On type-2 complexity classes: Preliminary report. pages 123–138, May 2001.
18. A. R. Meyer and P. C. Fischer. Computational speed-up by effective operators. *The Journal of Symbolic Logic*, 37:55–68, 1972.
19. A. Nerode. General topology and partial recursive functionals. *Talks Cornell Summ. Inst. Symb. Log., Cornell*, pages 247–251, 1957.
20. Dag Normann. *Recursive on the Countable Functionals*, volume 811 of *Lecture Notes in Mathematics*. Springer-Verlag, New York, 1980.
21. Piergiorgio Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science Publishing, North-Holland, Amsterdam, 1989.
22. Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967. First paperback edition published by MIT Press in 1987.
23. Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. The MIT Press, third edition, 1992. Original edition published by McGraw-Hill in 1967. First paperback edition published by MIT Press in 1987.
24. 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.
25. 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.
26. Anil Seth. Complexity theory of higher type functionals. Ph.d. dissertation, University of Bombay, 1994.
27. J.R. Shoenfield. The mathematical works of S.C. Kleene. *The Bulletin of Symbolic Logic*, 1(1):9–43, March 1995.
28. V.A. Uspenskii. On countable operations (Russian). *Doklady Akademii Nauk SSSR*, 103:773–776, 1955.
29. 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.

30. Klaus Wagner and Gerd Wechsung. *Computational Complexity*. Mathematics and its applications. D. Reidel Publishing Company, Dordrecht, 1985.

Appendix

We start with an intermediate theorem known as “Pseudo-Speed-up Theorem.” We customized the proofs for our needs. More original proofs can be found in any of [2–4, 6, 25, 29, 30]. Let f and g be two functions over \mathbf{N} . For convenience, let relation $f =^* g$ denote the case that $\overset{\infty}{\forall} x[f(x) = g(x)]$, i.e., for all but finitely many x such that $f(x) = g(x)$. Similarly, let $f <^* g$ denote the case that $\overset{\infty}{\forall} x[f(x) < g(x)]$,

Theorem 6 (Pseudo-Speed-up Theorem). *Let $r : \mathbf{N} \rightarrow \mathbf{N}$ be recursive. There exists a recursive function $f_r : \mathbf{N} \rightarrow \mathbf{N}$ 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 $r : \mathbf{N} \rightarrow \mathbf{N}$. 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) $\varphi_e : \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$,
- 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 speedupable 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 speed-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 speed-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: $C_{u,u}$, $C_{u,u+1}$, \dots , $C_{u,x-1}$, and
2. The cost of computing each of $\varphi_{s(e,u+1)}(x)$, $\varphi_{s(e,u+2)}(x)$, \dots , $\varphi_{s(e,x)}(x)$.

Figure 1 shows the dependence of $C_{u,x}$ on these previously defined 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 $\varphi_e(u, x) = 1 + \max(\{\varphi_i(x) \mid i \in C_{u,x}\})$, where

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

Claims:

1. φ_e is total.
2. For every $u, x \in \mathbf{N}$, $C_{u,x} = C_{0,x} \cap \{u, u + 1, \dots, x - 1\}$.

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 $x \in \mathbf{N}$, $\varphi_e(0, x) = \varphi_e(u, x)$.
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 , respectively. 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 $u \in \mathbf{N}$, $C_{u,x} = C_{0,x} \cap \{u, u+1, \dots, x-1\}$.

Inductive step: We argue that, when $x = n+1$, then $C_{u,x} = C_{0,x} \cap \{u, u+1, \dots, x-1\}$ 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,

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

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

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

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, $C_{0,x} - C_{u,x} \subseteq \{0, 1, \dots, u-1\}$. Thus, only indices in $\{0, 1, \dots, u-1\}$ 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 $\{0, 1, \dots, u-1\}$ 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 cancellation set $C_{0,a}$. Hence, $\varphi_i(a) \neq \varphi_{s(e,0)}(a)$. This is a contradiction.

This completes the proof of the Pseudo-Speed-up Theorem. \square

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 speedupable 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.⁵

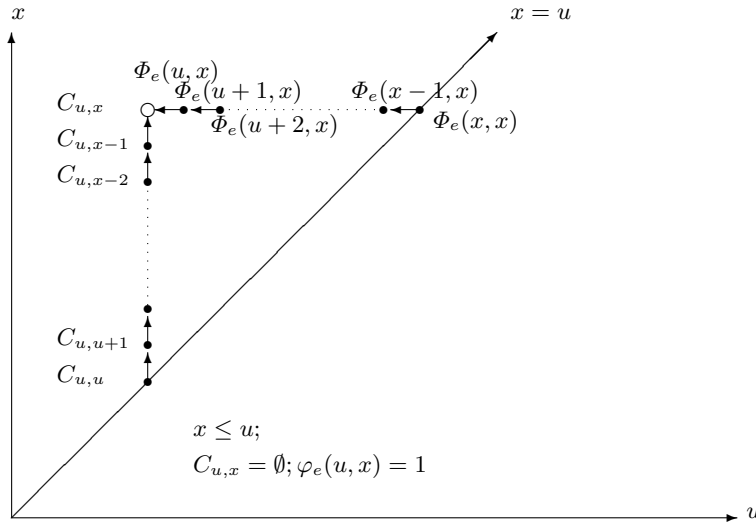


Fig. 1. The dependence of $C_{u,x}$ on previously defined sets and run times

Type-2 S-m-n and Recursion Theorems: The s-m-n theorem and the recursion theorem are essential tools in the study of computable functions. Let $\langle \cdot, \cdot \rangle : \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$ be a fixed pairing function (see [23], page 64), which is simply a computable bijection between $\mathbf{N} \times \mathbf{N}$ and \mathbf{N} .

After Kleene's [12, 14], many approaches have been proposed in order to extend the recursion theory to higher-type functionals.⁶ Following Kleene's no-

⁵ A rather comprehensive discussion about the constructibility of the proof of the Speed-up Theorem can be found in [29].

⁶ In Odifreddi [21], page 199, or Shoenfield [27], one can find a brief discussion about Kleene's work on the subject. For more details, see Normann [20] or Gandy and Hyland [8].

tations and his S1 - S9 of [12] (the inductive definition for the notion of higher-type computability), we can establish an s-m-n theorem for higher-type *countable* functionals in the following from:

$$\{e\}(\varphi_1, \dots, \varphi_n, \psi_1, \dots, \psi_m) = \{S(e, \varphi_1, \dots, \varphi_n)\}(\psi_1, \dots, \psi_m). \quad (3)$$

The underlying machines in Kleene's countable functionals are OTMs. Thus, at type-2, the s-m-n theorem in our notation is the following: There is a recursive function $s : \mathbf{N} \rightarrow \mathbf{N}$ such that, for every $f, g \in \mathcal{T}, x \in \mathbf{N}$, we have

$$\widehat{\varphi}_e(f, g, x) = \widehat{\varphi}_{s(e)}^f(g, x).$$

However, this does not directly help our work in the present paper. For obvious reasons, we cannot fix an arbitrary type-1 function as built-in data for a type-2 functional unless the type-1 function itself is computable. Thus, there is no computable $S : \mathbf{N} \times \mathcal{T} \rightarrow \mathbf{N}$ such that, for every $\widehat{\varphi}_e : \mathcal{T} \times \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}, f, g \in \mathcal{T}$, and $x \in \mathbf{N}$, we have

$$\widehat{\varphi}_{S(e,f)}(g, x) = \widehat{\varphi}_e(f, g, x).$$

On the other hand, by slightly modifying the proofs of the ordinary s-m-n and recursion theorems, we can establish restricted type-2 s-m-n and recursion theorems. We state the following two theorems with proof omitted. The two theorems are all we need in our construction.

Theorem 7 (Type-2 s-m-n theorem on type-0 argument). *There exists a recursive function $s : \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$ such that, for every $f \in \mathcal{T}$ and $e, x, y \in \mathbf{N}$, we have*

$$\widehat{\varphi}_{s(e,x)}(f, y) = \widehat{\varphi}_e(f, \langle x, y \rangle).$$

Theorem 8 (Type-2 recursion theorem). *There is a recursive function $r : \mathbf{N} \rightarrow \mathbf{N}$ such that, for every $f \in \mathcal{T}$ and $e, x \in \mathbf{N}$, we have*

$$\widehat{\varphi}_{r(e)}(f, x) = \widehat{\varphi}_e(f, \langle r(e), x \rangle).$$

The two theorems above are key tools in what follows.

(Theorem 4 Type-2 Pseudo-Speed-up Theorem) *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, for every $\widehat{\varphi}$ -program i for F_r , there is another $\widehat{\varphi}$ -program j such that,*

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

Fix a recursive function $r : \mathbf{N} \rightarrow \mathbf{N}$. With the s-m-n and recursion theorems on the type-0 argument introduced above, 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 6, such that:

- a) $\widehat{\varphi}_e : \mathbf{N} \times \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$.
- 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 cancellation 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:

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

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

$$\widehat{\varphi}_e(u, f, x) = 1 + \max(\{\widehat{\varphi}_i(f_0, x) \mid i \in C_{u,x}\}). \quad (4)$$

In addition to the five claims in the proof of Theorem 6, we add one more claim 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 $u, x \in \mathbf{N}$, $C_{u,x} = C_{0,x} \cap \{u, u+1, \dots, x-1\}$.
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

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

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

Claims 2, 3, 4, and 5 can be proven by exactly the same arguments for Theorem 6.

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 $\widehat{\varphi}_{s(e,0)}$. \square