Journal Version

Title:

Almost-everywhere Relations for Type-2 Algorithms and Compactness in Induced Baire Topologies

Author:

Chung-Chih Li, Ph.D.

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

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

e-mail: cli2@ilstu.edu

Note:

The preliminary version of the paper was presented in the 3^{rd} Conference of Theoretical Computer Science, jointed with IFIP's 18^{th} WCC in Toulouse, France, 2004. Here we present its full version with detailed proof of the theorems asserted.

Almost-everywhere Relations for Type-2 Algorithms and Compactness in Induced Baire Topologies

Chung-Chih Li

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

December 14, 2007

Abstract

The present paper is intended to propose an alternative notion of asymptotic behaviors for the study of type-2 computational complexity. Since the classical asymptotic notion (*for all but finitely many*) is not acceptable in type-2 context, we alter the notion of "small sets" from "finiteness" to topological "compactness" in order to establish a general type-2 complexity theory. A natural reference for type-2 computations is the standard Baire topology. We point out some serious drawbacks of using the standard Baire topology and then introduce an alternative topology for describing compact sets. Following our notion an explicit type-2 complexity class can be well-defined in terms of complexity (resource) bounds, and such a complexity class can be understood in the sense of the classical machine model. The bound can be either a type-1 function or a type-2 functionals since our complexity theorems are robust to either choice (we choose type-2 functionals in this paper for simplicity). In particular, we show that the complexity classes following our notion are recursively representable; namely, a programming system for each complexity class must exist. We also prove a type-2 analog of Rabin's theorem and a type-2 Gap theorem to provide evidence that our notion of type-2 almost-everywhere relations is workable. We speculate that our research will give rise to a possible approach in examining the complexity structure at type-2 along the line of the classical complexity theory.

Keywords: Type-2 Computation, Computational Complexity, Baire Topology, Asymptotic Notation.

1 Introduction

A key notion involved in defining the complexity of a problem is the use of "finiteness". We say that, function f is asymptotically bounded by g if and only if *for all but finitely many* $x \in \mathbf{N}$ such that $f(x) \leq g(x)$. Namely,

$$f \leq^* g \quad \Longleftrightarrow \quad \exists x_0 \in \mathbf{N} \forall x \in \mathbf{N}[x > x_0 \Rightarrow f(x) \leq g(x)]. \tag{1}$$

Based on the notion above, Hartmanis and Stearns [12] gave the very first precise definition for explicit complexity classes in the following form:

$$\mathbf{C}(t) = \left\{ \varphi_e \;\middle|\; \Phi_e \leq^* t \right\}, \tag{2}$$

where t is a computable function and $\langle \varphi_i \rangle_{i \in \mathbf{N}}$ is an *acceptable programming system* [25] with a *complexity measure* $\langle \Phi_i \rangle_{i \in \mathbf{N}}$ associated to it [2]. The use of $\leq^*$ can also be found elsewhere, e.g., the asymptotic notations in algorithm analysis, i.e., Θ, Ω, O, and so on. The most important consequence of using asymptotic notations, in our opinion, is not just that we can significantly simplify our notations, but that it is an indispensable tool in the theoretical study of computational complexity. Almost all nontrivial complexity theorems at the center of classical complexity theory such as the Speedup Theorem [2, 32], the Union Theorem [20], the Gap Theorem [4, 6, 32], the Compression Theorem [2], the Honesty Theorem [20], and so on (see [27] for more), are all proven with the use of a mathematical technique called *priority method* [29]. The method argues that the required properties or behaviors of a program being constructed will be obtained eventually in the process of the construction. For a more practical example, consider *recursive relatedness theorem* [2], which allows us to abstract away from any specific computing model and obtain a general complexity theory [23]. The theorem is proven also based on the use of asymptotic behaviors. It is also worth to note that the asymptotic notation is not arbitrary just for convenience. It is justified by the fact that a program can be patched on some fixed finitely many inputs by using a finite table in which we directly code the values of the function to avoid some expensive computations. Thus, theoretically, we can use $\leq$ and $\leq^*$ in (2) alternatively without changing the underlying complexity property of computable functions since we can linearly speedup[1].

When we shift our attention to higher ordered computation, which in many cases seems to be a better computing model for many contemporary computing problems, we soon realize that there is no general complexity theory to unify results from different approaches. In our opinion, the main reason is that we do not have a reasonable notion of higher ordered asymptotic behaviors as the one involved in (2). The direct use of "for all but finitely many" in type-2 computation is not acceptable, because we cannot patch a type-2 program on a function input in general. Consequently, many techniques used in the proofs of classical complexity theorems are not applicable in type-2 context. Therefore, the present paper is intended to provide a robust notion of type-2 asymptotic behaviors so that the classical complexity theory can be advanced into type-2. Since any machine model for computation beyond type-2 seems inconceivable by intuitions, we thus focus on type-2 computation which can be intuitively modelled by the antiquity – Oracle Turing Machine (see [10] for conventions).

Notations: A type-0 object is simply a natural number. A type-1 object is a function over natural numbers. A type-2 object is a *functional* that takes and produces type-1 objects. By

[1] Precisely speaking, this is an overstatement, since we have overlooked the honesty property of t. The bound, t, needs to be honest so we can finitely path the program under t. Nevertheless, we take the liberty to drop the honesty condition since it is a rather weak condition that most natural resource bounds have no problem to bear.

© Chung-Chih Li

convention, we consider type-0 $\subset$ type-1 $\subset$ type-2. We are only interested in total functions, $\mathbf{N} \to \mathbf{N}$, when they are taken as inputs of functionals. For convenience, we use $\mathcal{T}$ to denote the set of total functions and $\mathcal{P}$ to denote the set of partial functions. Note that functions in $\mathcal{T}$ or $\mathcal{P}$ may not be computable. Also, we use $\mathcal{F}$ to denote the set of *finite* functions, which means $\sigma \in \mathcal{F}$ if and only if $\mathsf{dom}(\sigma) \subset \mathbf{N}$ and $\mathsf{card}(\sigma) \in \mathbf{N}$. We fix a canonical indexing for $\mathcal{F}$, and hence we are free to treat any function in $\mathcal{F}$ as a number so it can be taken as the input of a type-1 function. Unless stated otherwise, we let a, b, x, y, z range over $\mathbf{N}$, f, g, h range over $\mathcal{T}$, and F, G, H range over type-2 functionals. Here we consider some examples of type-2 functionals:

1. $F : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}, \quad F(f, x) = f(x)$.
2. $G : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}, \quad G(f, x) = f(f(x))$.
3. $H : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}, \quad H(f, x) = \sum_{i=0}^{x} f(i)$.
4. $\Gamma : \mathcal{T} \rightharpoonup \mathcal{T}, \quad \Gamma(f) = f \circ f$ (function composition).

Clearly, F, G, and H are type-2 functionals of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$, and Γ is a type-2 functional of type $\mathcal{T} \to \mathcal{T}$. With λ-abstraction, we have $\Gamma(f) = \lambda x G(f, x)$. Since some complexity properties at type-2 can be easily proven by the same tricks used in the original proofs, we therefore keep a type-0 input in order to take this advantage. We also note that $\mathcal{T} \cong \mathcal{T} \times \mathbf{N}$ via, for example, $f \mapsto (f', f(0))$, where $f'(x) = f(x+1)$. Thus, we do not lose generality when we restrict type-2 functionals to our standard type $\mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$.

Although the type-1 input itself is an infinite object in general, we observe that only a finite part of it is needed for any terminating computation. This is a trivial application of the following theorem due to Uspenskii [31] and Nerode [21]: *A functional F is continuous if and only if F is compact and monotone.* Compactness and monotonicity are defined as follows.

Definition 1 *Let $F : (\mathbf{N} \rightharpoonup \mathbf{N}) \times \mathbf{N} \rightharpoonup \mathbf{N}$. We say that:*
*(i) F is **compact** if and only if*

$$\forall (f, x) \in (\mathbf{N} \rightharpoonup \mathbf{N}) \times \mathbf{N} \; \exists \sigma \in \mathcal{F} \; \left[F(f, x) = F(\sigma, x) \right].$$

*(ii) F is **monotone** if and only if*

$$\forall (\sigma, x) \in \mathcal{F} \times \mathbf{N} \; \left[F(\sigma, x) \downarrow \Rightarrow \; \forall \tau \supseteq \sigma (F(\tau, x) \downarrow = F(\sigma, x)) \right].$$

Compactness and monotonicity will become the key properties of computable functionals in defining our topologies. We take Oracle Turing Machines (OTM here after) as our formal type-2 computing device, where the oracle is extended from a set-oracle to a function-oracle. Thus, by a *computable functional*, we mean a functional that can be computed by some OTM, where the type-1 input will be prepared as an oracle attached to the machine.

Clearly, every computable functional is continuous [25]. As for classical Turing Machines, we can fix a programming system $\langle \widehat{\varphi} \rangle_{i \in \mathbf{N}}$ associated with a complexity measure $\langle \widehat{\Phi} \rangle_{i \in \mathbf{N}}$ for our OTM's. Conventionally, we take the number of steps an OTM performed as

 © Chung-Chih Li

our complexity measure. Clearly, Blum's two axioms can be used directly without any modification. However, having Blum's axioms for type-2 complexity measures does not mean a general complexity theory immediately follows. A workable notion of type-2 asymptotic behaviors indeed is the missing piece of the current type-2 complexity theory.

2 An Outlook of Present Complexity Theory at Type-2

2.1 Basic Feasible Functionals

The class of Type-2 Basic Feasible Functionals [8] (BFF here after) to some degree is seen as the type-2 analog of **P**. In [9, 13, 14] Cook and Kapron define *second-order polynomials* in order to characterize BFF. Their framework requires a rather artificial function called *length function*. The length function serves as the type-2 analogy of $|x|$, where $|x|$ is the length of the bit string representing $x \in \mathbf{N}$. Given any function $f \in \mathcal{T}$, the length of f is defined as a function by

$$|f| = \lambda n.\mathsf{max}(\{\ell \ : \ \ell = |f(x)| \text{ and } |x| \leq n\}).$$

We say that $|f|$ is the length function of f. For $n \in \mathbf{N}$, $|f|(n)$ is the maximum length of the values of f on inputs with length $\leq n$. Kapron and Cook obtain a very neat theorem helping BFF to be understood in the way we understand **P**. We state the theorem in the following.

Theorem 2 (B. Kapron and S. Cook [13, 14]) *A type-2 functional F is a BFF if and only if there is an OTM for F and a second-ordered polynomial $\boldsymbol{p}$ such that, on every $f \in \mathcal{T}$ and $x \in \mathbf{N}$, the run time of the OTM is bounded by $\boldsymbol{p}(|f|, |x|)$.*

Is it conceivable to use second-ordered polynomials and length functions to further classify functionals inside the class of BFF? We present an easy example to show that a naive treatment can easily lead to a result drifting away from our intuition. Consider the following two BFF's of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$:

$$\begin{aligned} F(f,x) = \ & 2^{\mathsf{max}\{f(2^{|x|}),f(2^{|x|}-1),\ldots,f(2^{|x|}-(|x|-1))\}}; \\ G(f,x) = \ & \begin{cases} 2^{2^{10}} & \text{if } x = 0 \text{ and } f(2^{|x|}) = 0; \\ 2^{\mathsf{min}\{f(2^{|x|}),f(2^{|x|}-1),\ldots,f(2^{|x|}-(|x|-1))\}} & \text{otherwise.} \end{cases} \end{aligned} \tag{3}$$

Let $\widehat{\Phi}_F$ and $\widehat{\Phi}_G$ denote the cost functions of F and G, respectively, e.g., the number of steps an OTM performed. Clearly, a bigger query requires more time to prepare. We observe that the major cost of computing the two functionals in (3) is the cost of querying the oracle. For each query q, at least $O(|q|+|f(q)|)$ steps are needed; where $O(|q|)$ steps are for placing the query and $O(|f(q)|)$ for reading the answer returned from the oracle. Moreover, each

 © Chung-Chih Li

functional in (3) needs $|x|$ many queries and each query will obtain $|f(2^{|x|}-i)|$ many bits to read for some $i < |x|$. Thus, in term of length functions, both are bounded by

$$|x| \times O(|x| + |f|(|x|)) = O(|x|^2 + |x| \cdot |f|(|x|)).$$

Namely, both are bounded by the second-ordered polynomial, $\boldsymbol{p}(\ell, x) = c(x^2 + x \cdot \ell(x))$ for some $c \in \mathbf{N}$. However, we also observe that, unless $x = f(2^{|x|}) = 0$ and the cost of querying $f(2^{|x|}), f(2^{|x|}-1) \cdots f(2^{|x|}-(|x|-1))$ are small compared to the cost of printing $2^{2^{1000}}$ as output, have we $\widehat{\Phi}_G(f,x) \leq \widehat{\Phi}_F(f,x)$. In other words, in *most* cases, we have $\widehat{\Phi}_G(f,x) \leq \widehat{\Phi}_F(f,x)$. The problem is, we have difficulty to formally describe this quite simple situation. Not to mention the rich class of type-2 functionals beyond BFF, the second-ordered polynomial has its limitation to further tighten complexity bounds for separation due to the use of the length function. What should be a formal and satisfactory notion of "most" cases? How do we formalize the concept of "*most*" cases so that we can forgive a "*few*" exceptional cases that are "*affordable*"? On what ground we can justify our intuition that G is easier than F? We shall answer these question in this paper.

2.2 Where We Head For

It is unfortunate that the study of type-2 complexity beyond BFF is almost empty[2]. For a general type-2 complexity to begin with, arguably but certainly a reasonable direction, we need to have a robust notion of type-2 complexity classes along the line of Hartmanis and Stearns' definition as shown in (2). Here we consider Kapron and Cook's setting again as their work currently seems to be the most suitable framework for the study of type-2 complexity classes in terms of explicit bounds.[3] A type-2 complexity class determined by a second-order polynomial $\boldsymbol{p}$ can be formulated as follows,

$$\mathbf{C}(\boldsymbol{p}) = \left\{ \widehat{\varphi}_e \;\middle|\; \forall (f,x) \in \mathcal{T} \times \mathbf{N} \left[\widehat{\Phi}_e(f,x) \leq \boldsymbol{p}(|f|,|x|) \right] \right\}. \tag{4}$$

In [28] Seth also suggested a type-2 complexity class similar to (4) where $\boldsymbol{p}$ was extended to any type-2 computable functional. Seth speculated that we may be able to prove some classical complexity results such as the Gap theorem and the Union theorem. However, we are skeptical about this because we notice that there is no notion of asymptotic behaviors involved in (4) and, as we mentioned earlier, the original proofs of the two theorems among others rely on *priority arguments* in which some violations need to be tolerated. In fact, we suspect that it is impossible to prove any nontrivial complexity theorems without such tolerance.

An immediate idea is to keep the same notion of $\leq^*$ and implant it in (4) directly. However, this is problematic, because there is no corresponding Church-Turing thesis at type-2. In other words, there is no effective way to patch a program on finitely many type-1

[2]There are some rather old investigations initiated by some luminaries such as S. Kleene, R. Constable, et al. [15, 16, 11, 7]. But after their works, no significant development follows.

[3]Another line for the study of higher-order complexity theory is *Implicit Computational Complexity*; no explicit resource bounds are used to name higher-order complexity classes.

© Chung-Chih Li

inputs simply because we cannot build the entire body of an arbitrary type-1 function in a program (the function may not be computable).

Another way to get around the problem is to consider only "seen computation". As a matter of fact, all terminating computations are finite and countable, and hence enumerable. Based on this observation, in early 70's Symes [30] presented a work on type-2 complexity theory with an axiomatic approach. The axiomatic system was modified from Blum's. Symes required a computation (represented by a computational tree) of the concerned type-2 functionals to be explicitly provided as an input. The computing machine will be shut down if the provided computation is not consistent with the "real" computation of the machine. In his proofs, $\leq^*$ was used in the context as "for all but finitely many computations". This seems to be a reasonable setup in a sense that, for every computable type-2 functionals F and G, we consider F almost-everywhere less than G (i.e., $F \leq^* G$) if there are only finitely many computations of F and G resulting in $F > G$. However, the setup is too remote to be practical. What we can do is to enumerate all possible computations only for theoretical investigation.

In the following sections we introduce a new idea in defining type-2 almost-everywhere relation, $\leq_2^*$. We also provide evidence to show that our notion of $\leq_2^*$ is workable looked at both practical and theoretical aspects. As "compact" used in topology to some extent is considered as a surrogate for "finite", "small", and "computable", our investigation begins with a study on the close relation between topology and type-2 computation.

3 Topologies and Type-2 Computation

Notations: Recall that $\mathcal{T}$ is the set of total functions of type $\mathbf{N} \to \mathbf{N}$ and $\mathcal{F}$ the set of finite functions over $\mathbf{N}$. Let $\mathbb{N}$ be the discrete topology on $\mathbf{N}$. The space $\mathcal{T}$ is called Baire space. Let the Baire topology [1, 22, 25] be denoted by $\mathbb{T}$. A basic open set of $\mathbb{T}$ is the set of all total extensions of some finite function. Since we restrict our type-1 inputs to total functions, it is very natural to have our attention on the Baire topology. Moreover, since the only type considered here is $\mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$, we are interested in the product topology $\mathbb{T} \times \mathbb{N}$. Given $F, G : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$, we use $X_{[F \leq G]} \subseteq \mathcal{T} \times \mathbf{N}$ to denote the set of all (f, x) such that $F(f, x) \leq G(f, x)$. Similarly, we have $X_{[F=X]}$ and $X_{[F>G]}$ defined in the same manner. A type-2 functional F is said to be computable if there is an OTM with index e that computes F, i.e., $F = \widehat{\varphi}_e$.

3.1 Intuitive Almost-Every Relations and Their Problems at Type-2

For $F, G : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$, a straightforward analog of $F \leq^* G$ would be:

$$\textit{"For all but finitely many } (f, x),\ F(f, x) \leq G(f, x)\textit{."} \tag{5}$$

However, as we pointed out earlier, (5) is too restrictive, because in general there is no terminating computation that can recognize finitely many (f, x)'s. Thus, if we are interested in computable functionals, a better notion of $F \leq^* G$ would be something like: *"For all but*

 © Chung-Chih Li

finitely many computations of F and G, the result of F is less than or equal to the result of G", which is simply Symes' proposal in [30]. We would put a subscript "$_{\mathbf{2}}$" in $\leq_2^*$ to reflect the type of its operands for the time being. Also, we would use "computation" to mean "terminating computation". A computation of a computable functional is simply a branch with finite length of its computation tree. We first state two naive objectives for an *ideal* type-2 almost-everywhere relation to consider.

Goal 1: $F \leq_2^* G$ if and only if there are two OTM's for F and G, respectively, such that, there are only finitely many computations of the two OTM's resulting in values such that $F > G$.

Goal 2: The type-2 relation $\leq_2^*$ should be *transitive.*

In type-1, the two objectives are rather trivial. Nevertheless, the first one assures us that we can patch a program, and the second one assures us that we do not lose any functions from a complexity class by increasing the resource bound. For the obvious reason, we want to preserve the two properties in defining type-2 complexity classes. Unfortunately, the two objectives conflicts; they hurt each other. In the end, we give up transitivity in order to achieve our primary purpose: a workable notion of type-2 asymptotic behaviors for a general type-2 complexity theory. The following standard theorem hints a possible way to formalize "$\leq_2^*$".

Theorem 3 *Let $\widehat{\varphi}_e$ be total and let $S \subset \mathcal{T} \times \mathbf{N}$. If S is compact in $\mathbb{T} \times \mathbb{N}$, then there are only finitely many computations of $\widehat{\varphi}_e$ on S.* □

Thus, it seems reasonable to formalize our notion as follows: $F \leq_2^+ G$ if and only if there is a set X, such that X is $(\mathbb{T} \times \mathbb{N})$-compact, and for all $(f, x) \notin X$ we have $F(f, x) \leq G(f, x)$. Formally, we restate this in the following definition.

Definition 4 *Let $F, G : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$. $F \leq_2^+ G$ if and only if $X_{[F \leq G]}$ is co-compact in $\mathbb{T} \times \mathbb{N}$.*

The superscript, "$^+$", reflects the conclusion we will discuss in a moment that this definition is too strong for our purposes. We at first prove the following two standard lemmas.

Lemma 5 *Every open set is closed in $\mathbb{T} \times \mathbb{N}$.*

Proof: Fix any open set $S = \bigcup_{i \geq 0} (\!(\sigma_i, x_i)\!)$, where for every $i \geq 0$, $\sigma_i \in \mathcal{F}$ and $x_i \in N$. Suppose that $(f, x) \in \mathcal{T} \times \mathbf{N}$ and $(f, x) \notin S$. Thus, there exists $a \in \mathbf{N}$ such that, for every $i \geq 0$, $x \neq x_i$ or $a \in \mathsf{dom}(\sigma_i)$ and $\sigma_i(a) \neq f(a)$. Let $\tau = \{(a, f(a))\}$. Then, $(\!(\tau, x)\!)$ is an open set that contains (f, x) and $S \cap (\!(\tau, x)\!) = \emptyset$. Thus, (f, x) is not an accumulation point of S. By contrapositive, all accumulation points of S are in S. Therefore, S is closed. □

Lemma 6 *Let $S \subseteq X$. If X is compact in $\mathbb{T} \times \mathbb{N}$, then so is S.*

© Chung-Chih Li

Proof: Let $X \subseteq \mathcal{T} \times \mathbf{N}$ be compact in $\mathbb{T} \times \mathbb{N}$, and $S \subseteq X$. By contradiction, suppose that S is open. Consider the standard result of the Baire topology that every nonempty open set is not compact in $\mathbb{T} \times \mathbb{N}$. It follows that there is an open cover for S without any finite subcover. Let $\mathcal{O}$ by such an open cover. By Lemma 5, S is closed. Thus, S^c, the complement of S, is open, and hence $\{S^c\} \cup \mathcal{O}$ is an open cover for X without a finite subcover. This contradicts the assumption that X is compact. □

The following theorem shows that the relation $\leq_2^+$ meets our Goal 2.

Theorem 7 *The relation, $\leq_2^+$, is transitive over type-2 continuous functionals.*

Proof: Let $F, G, H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be continuous. Suppose $F \leq_2^+ G$ and $G \leq_2^+ H$. By definition, $X_{[F>G]}$ and $X_{[G>H]}$ are compact in $\mathbb{T} \times \mathbb{N}$. Clearly, $X_{[F>G]} \cup X_{[G>H]}$ is also compact. Since

$$X_{[F>H]} \subseteq X_{[F>G]} \cup X_{[G>H]},$$

by Lemma 6, it follows that $X_{[F>H]}$ is also compact. Therefore, $F \leq_2^+ H$. □

Also, consider the following standard property of Baire topology: if S is compact, then the image of any continuous functional on S is also compact. Also, every compact set in $\mathbb{N}$ is finite. We have the following corollary.

Corollary 8 *If F and G are continuous and $F \leq_2^+ G$, then there exists $c \in \mathbf{N}$ such that, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, $F(f, x) \leq G(f, x) + c$.*

Proof: Suppose F and G are continuous and $F \leq_2^+ G$. By definition, the set $X_{[F>G]}$ is compact in $\mathbb{T} \times \mathbb{N}$. Define $H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ by

$$H(f, x) = \mathsf{max}\left(\left\{F(f, x) - G(f, x), 0\right\}\right).$$

Since F and G are continuous, it follows that H is also continuous, and hence $H(X_{[F>G]})$ is compact in $\mathbb{N}$. Since any compact set of the discrete topology $\mathbb{N}$ is finite, $H(X_{[F>G]})$ is bounded by some $c \in \mathbf{N}$. The theorem follows immediately. □

The similarity between $f \leq^* g$ and $F \leq_2^+ G$ can be easily seen. We display (1) again for comparison in the following.

- Type-1: $f \leq^* g$ if and only if

$$\exists x_0 \in \mathbf{N} \forall x \in \mathbf{N} \left(x > x_0 \Rightarrow f(x) \leq g(x)\right);$$

- Type-2: $F \leq_2^+ G$ if and only if

$$\exists f_0 \in \mathcal{T} \exists x_0 \in \mathbf{N} \forall f \in \mathcal{T} \forall x \in \mathbf{N} \left((f > f_0) \vee (x > x_0) \Rightarrow F(f, x) \leq G(f, x)\right).$$

© Chung-Chih Li

This suggests that Definition 4 might be a right choice. Also, Corollary 8 shows that if $F \leq_2^+ G$, then by adding some constant value c to G, F will be bounded everywhere. If we consider G as some sort of resource bound, we speculate that a constant or *linear speedup theorem* may be proven. And, if this is the case, we can as well modify the program for F to remove the extra constant cost c. However, $\leq_2^+$ has a formidable problem discouraging us to move any further. We show that the flexibility introduced by $\leq_2^+$ in fact is an empty notion; we hardly gain any benefit from it to prove any nontrivial theorems. Consider the following theorem.

Theorem 9 *Let $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be continuous. $F \leq_2^+ G$ if and only if $X_{[F>G]} = \emptyset$.*

Sketch of Proof: The proof is an application of the Uspenskii-Nerode theorem that every continuous functional must be *compact*. Thus, if $X_{[F>G]}$ is not empty, it must be $(\mathbb{T} \times \mathbb{N})$-open. But the only set in $\mathbb{T} \times \mathbb{N}$ that is both open and compact is the empty set. Therefore, if $X_{[F>G]}$ is not empty, it can't be compact in $\mathbb{T} \times \mathbb{N}$. □

To fix this problem, we will need a topology that is coarser than the standard Baire topology. In such a topology, the compact sets are abundant enough for us to describe "small" sets.

4 Type-2 Almost-Everywhere Relations

In this section we define a class of topologies determined by the functionals involved in the relations. These topologies are induced from the Baire topology. On the one hand, the induced topology must be coarse enough so that the compact sets are not necessarily trivial. On the other hand, the induced topology must be fine enough so that we can differentiate two computations in terms of their computations depending on their type-1 inputs. Also, the formalization of the almost-everywhere relation should catch the intuitive idea stated in previous sections. Unfortunately, the two goals proposed in Section 3 are difficult to achieve at the same time. We are forced to give up our intuitive idea about transitivity, which we trade in for a workable notion of type-2 asymptotic behaviors.

Given $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, let $F(f, x) \downarrow = y$ denote the case that F is defined on (f, x) and its value is y. Consider the following definitions.

Definition 10 *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.* □

© Chung-Chih Li

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 convenience, we use $(\!(\sigma)\!)$ to denote the set of total extensions for any $\sigma \in \mathcal{F}$, i.e., $(\!(\sigma)\!) = \{f \in \mathcal{T} \mid \sigma \subset f\}$. We extend this notation to $(\!(\sigma, x)\!) = \{(f, x) \mid f \in (\!(\sigma)\!)\}$. For each $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, we take $(\!(\sigma, x)\!)$ as a basic open set of $\mathbb{T} \times \mathbb{N}$. We observe that, for every $\sigma_1, \sigma_2 \in \mathcal{F}$ and $x_1, x_2 \in \mathbf{N}$,

$$(\!(\sigma_1, x_1)\!) \cap (\!(\sigma_2, x_2)\!) = \begin{cases} \emptyset & \text{if } x_1 \neq x_2; \\ \big[(\!(\sigma_1)\!) \cap (\!(\sigma_2)\!)\big] \times \{x_1\} & \text{if } x_1 = x_2. \end{cases}$$

Note 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 $(\!(\sigma_1, x_1)\!) \cup (\!(\sigma_2, x_2)\!)$ is conventional and an arbitrary union may result in an open set that is not basic. Given any $f, g \in \mathcal{T}$ and $a \in \mathbf{N}$, 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 $\sigma(k) \neq \tau(k)$. Namely, $\mathbb{T} \times \mathbb{N}$ is a Hausdorff (T_2) topology on $\mathcal{T} \times \mathbf{N}$.

4.1 The Induced Topology $\mathbb{T}(F_1, F_2, \ldots, F_n)$ on $\mathcal{T} \times \mathbf{N}$

In stead of taking every $(\!(\sigma, x)\!)$ as a basic open set (this will form the Baire topology), we consider only those that are related to the concerned functionals. We introduce a class of relative topologies determined by some interested functionals.

Definition 11 *Given any finite number of continuous functionals, $F_1, F_2, \ldots, F_n$, let $\mathbb{T}(F_1, F_2, \ldots, F_n)$ denote the topology induced from $\mathbb{T} \times \mathbb{N}$ by functionals $F_1, F_2, \ldots, F_n$, where the basic open sets are defined as follows: For each $(f, a) \in \mathcal{T} \times \mathbf{N}$ and $1 \leq i \leq n$, let (σ_i, a) be the minimal locking fragment of F_i on (f, a). Let*

$$(\!(\sigma, a)\!) = (\!(\sigma_1, a)\!) \cap (\!(\sigma_2, a)\!) \cap \cdots \cap (\!(\sigma_n, a)\!). \tag{6}$$

Then, each $(\!(\sigma, a)\!)$ is taken as a basic open set of $\mathbb{T}(F_1, F_2, \ldots, F_n)$.

Note that, in the definition above, we have $\sigma = \bigcup_{1 \leq i \leq n} \sigma_i$. Thus, if $(\!(\sigma, a)\!)$ is a basic open set of $\mathbb{T}(F_1, F_2, \ldots, F_n)$, then (σ, a) must be a locking fragment of $F_1, F_2, \ldots, F_n$. However, given any two functionals, F_1 and F_2, the topologies $\mathbb{T}(F_1)$ and $\mathbb{T}(F_2)$ are not related, and hence the compactness cannot be transferred between the two topologies. This in fact is the inherited difficulty of having a transitive relation. We will discuss this in detail later. (Also, see footnote 4.)

4.2 Type-2 Almost-Everywhere Relation, $\leq_2^*$

Now, we are in a position to define our type-2 almost-everywhere relation.

Definition 12 *Let $F_1, F_2 : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be continuous. Define*

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

 © Chung-Chih Li

The complement of $X_{[F_1 \leq F_2]}$ is $X_{[F_1 > F_2]}$. We call set $X_{[F_1 > F_2]}$ the *exceptional set* of $F_1 \leq_2^* F_2$.

Theorem 13 *There are two computable functionals $F_1, F_2 : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that, $F_1 \leq_2^* F_2$ and, for any two OTM's that computes F_1 and F_2, respectively, there are infinitely many computations resulting in $F_1 > F_2$.*

Sketch of Proof: We simply observe that, given any two computable functionals F_1 and F_2, F_2 may not be continuous in topology $\mathbb{T}(F_1)$. Thus, $X_{[F_1 > F_2]}$ being compact in $\mathbb{T}(F_1)$ does not mean $F_2(X_{[F_1 > F_2]})$ must be compact in $\mathbb{T}(F_1)$. Also, $X_{[F_1 > F_2]}$ may not be compact in $\mathbb{T}(F_2)$, and hence $F_2(X_{[F_1 > F_2]})$ is not necessarily compact in $\mathbb{T}(F_2)$. □

Clearly, our Goal 1 fails.[4] Nevertheless, we feel that the statement in Goal 1 is too strong in the context of type-2. Our real intension behind Goal 1 is to be able to patch a program. We have the following theorem that fulfills our real purpose behind Goal 1.

Theorem 14 *Suppose $F_1, F_2 : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ are computable. If $F_1 \leq_2^* F_2$, then there is an OTM for F_1 such that, there are only finitely many computations of the OTM on $X_{[F_1 > F_2]}$.*

Proof: If $X_{[F_1 > F_2]}$ is compact in $\mathbb{T}(F_1)$, we can fix a finite open cover for $X_{[F_1 > F_2]}$. Let this cover be $\{(\!(\sigma_1, x_1)\!), \ldots, (\!(\sigma_n, x_n)\!)\}$, where $n \in \mathbf{N}$. By the definition of $\mathbb{T}(F_1)$, each such (σ_i, x_i) is a minimal locking fragment of F_1. Thus, we can construct an OTM for F_1 such that, on any input $(f, x) \in X_{[F_1 > F_2]}$, the OTM does not query the f-oracle beyond $\mathsf{dom}(\sigma_1 \cup \cdots \cup \sigma_n)$. There are only finitely many possible answers to be returned from the oracle. □

The theorem above assures that there is a way to patch a $\widehat{\varphi}$-program on a compact set in the concerned topology. As for transitivity, we have a negative result shown by an easy example as follows.

Theorem 15 *The relation $\leq_2^*$ is not transitive.*

Proof: Suppose $F_1 \leq_2^* F_2$ and $F_2 \leq_2^* F_3$. The idea is that, since the topologies $\mathbb{T}(F_1)$ and $\mathbb{T}(F_2)$ are independent, we cannot guarantee that any subset of $X_{[F_2 > F_3]}$ is compact in

[4] We could have given Definition 12 as

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

This definition gives us a finer relative topology to assure that every involved functional is also continuous in $\mathbb{T}(F_1, F_2)$. It follows that we can have a theorem opposite to Theorem 13 and achieve our Goal 1. However, the break of Goal 1 due to the infinitely many computations of F_2 is acceptable, since F_2 mostly serves as a mathematical bound and its computation is not interested at all. Besides, the topology $\mathbb{T}(F_1, F_2)$ is still not fine enough to bring back transitivity to our type-2 almost everywhere relation. Therefore, we do not find any particular advantage of using $\mathbb{T}(F_1, F_2)$ as our reference topology for the compactness of $X_{[F_1 > F_2]}$.

© Chung-Chih Li

$\mathbb{T}(F_1)$. Thus, the set $X_{[F_1>F_3]} \subseteq X_{[F_1>F_2]} \cup X_{[F_2>F_3]}$ may not be compact in $\mathbb{T}(F_1)$. Consider the following example.

$$F_1(f,x) = \begin{cases} f(1) \texttt{ mod } 2 & \text{if } f(0)=0 \text{ and } x=0, \\ 0 & \text{otherwise.} \end{cases}$$

$$F_2(f,x) = \begin{cases} 2 & \text{if } f(0)=0 \text{ and } x=0, \\ 1 & \text{otherwise.} \end{cases}$$

$$F_3(f,x) = \begin{cases} f(1) \texttt{ mod } 3 & \text{if } f(0)=0 \text{ and } x=0, \\ 2 & \text{otherwise.} \end{cases}$$

It is clear that $X_{[F_1>F_2]} = \emptyset$, and hence $F_1 \leq_2^* F_2$. We observe that

$$X_{[F_2>F_3]} = \{(f,0) \mid f(0)=0 \text{ and } f(1) \leq 1\}.$$

Since every $(f,x) \in X_{[F_2>F_3]}$ must be in $(\!(\sigma,0)\!)$, where $\mathsf{dom}(\sigma) = \{0\}$ and $\sigma(0)=0$, which is the only basic open set of $\mathbb{T}(F_2)$ that contains such (f,x). Thus, any open cover for $X_{[F_2>F_3]}$ must have a set includes $(\!(\sigma,0)\!)$. Therefore, $X_{[F_2>F_3]}$ is compact in $\mathbb{T}(F_2)$, and hence $F_2 \leq_2^* F_3$. Now, consider

$$X_{[F_1>F_3]} = \{(f,0) \mid f(0)=0 \text{ and } f(1) \leq 3k \text{ for some } k \in \mathbf{N} \text{ with } k \geq 1\}.$$

Let $\mathsf{dom}(\sigma_i) = \{0,1\}$ and $\sigma_i(0)=0, \sigma_i(1)=i$. For every $i \in \mathbf{N}$, $(\!(\sigma_i,0)\!)$ is a basic open set of $\mathbb{T}(F_1)$. Let

$$\mathcal{O} = \{(\!(\sigma_3,0)\!), (\!(\sigma_6,0)\!), (\!(\sigma_9,0)\!), (\!(\sigma_{12},0)\!), \ldots\}.$$

Since $\mathcal{O}$ is an open cover for $X_{[F_1>F_3]}$ without finite subcover, it follows that $X_{[F_1>F_3]}$ is not compact in $\mathbb{T}(F_1)$. Therefore, $F_1 \not\leq_2^* F_3$ □

5 Applications in Type-2 Complexity Theory

Recall the two functionals F and G defined in (3). As for computational complexity, we will simply compare $\widehat{\Phi}_F$ and $\widehat{\Phi}_G$. Assume $|0|=1$ and hence $2^{|0|}=2$. Let $S = \{(\!(f,0)\!) \mid f(2) = 0\}$. We observe that, $X_{[\widehat{\Phi}_G>\widehat{\Phi}_F]} \subset S$. Since S is compact in $\mathbb{T}(\widehat{\Phi}_G)$[5], it follows that $X_{[\widehat{\Phi}_G>\widehat{\Phi}_F]}$ is also compact in $\mathbb{T}(\widehat{\Phi}_G)$. Therefore, $\widehat{\Phi}_G \leq_2^* \widehat{\Phi}_F$, which indeed reflects earlier intuitive comments that G is computationally easier than F.

In the following, we provide some serious applications of asymptotic behaviors of type-2 functionals due to our notion of type-2 almost-everywhere relations. We show that the

[5]Note that the topology $\mathbb{T}(\widehat{\Phi}_G)$ is determined by the locking fragments of the functional G, but not by the actual queries made during the course of the computation of G. There may be some unnecessary queries made. However, an optimal program should not make unnecessary queries just as an ordinary optimal type-1 program that should not go into some unnecessary loop.

© Chung-Chih Li

set of type-2 computable functionals asymptotically bounded by any given computable type-2 functional is recursively enumerable. In other words, every type-2 complexity class asymptotically bounded by some computable type-2 functional as resource bound has a programming system. Also, we prove a few complexity theorems at type-2 to show that the techniques used in classical complexity theory can be transferred to type-2 under our notion.

5.1 Type-2 Complexity Classes

In [19, 18] we define a special class of type-1 computable functions of type $\mathcal{F}\times\mathbf{N}\to\mathbf{N}$ called *Type-2 Time Bounds*. Under some proper *clocking scheme*, we give a type-2 complexity class $\mathbf{C}(\beta)$ determined by Type-2 Time Bound β. Since each Type-2 Time Bound β also determines a *limit functional* F_β, we can understand the complexity class $\mathbf{C}(\beta)$ by the following formula .

$$F\in\mathbf{C}(\beta)\Longrightarrow\exists e\left[\widehat{\varphi}_e=F\ \wedge\ \widehat{\Phi}_e\leq_2^* F_\beta\right]. \tag{7}$$

Note that, for every $\widehat{\varphi}$-program e, $\mathbb{T}(\widehat{\varphi}_e)=\mathbb{T}(\widehat{\Phi}_e)$. Since the way we clock an OTM not only depends on the result of F_β but also on the course of computing F_β, we do not have the converse of (7) in general. The complexity class $\mathbf{C}(\beta)$ is very sensitive to the clocking scheme and the conventions made for our OTM's. We may want to get rid of the specific knowledge of the clocking scheme in defining complexity classes. In the following, we give a more direct way in defining a type-2 complexity class, where the computable type-2 functional simply serves as the resource bound.

Definition 16 *Let $T:\mathcal{T}\times\mathbf{N}\to\mathbf{N}$ be computable. Define*

$$\mathbf{C}(T)=\left\{F\mid\exists e\left[\widehat{\varphi}_e=F\ \wedge\ \widehat{\Phi}_e\leq_2^* T\right]\right\}.$$

It is clear that the two complexity classes, $\mathbf{C}(T)$ and $\mathbf{C}(\beta)$, are not equivalent. Nevertheless, we speculate that the type-2 almost everywhere relation involved in Definition 16 will let us directly modify the proofs given in [18] such as the proofs of type-2 Speedup Theorem, Gap Theorem, Union Theorem, Compression Theorem, and so on. Also, we can directly modify the classical big-O notation for type-2 algorithms as follows:

Definition 17 *Let $T:\mathcal{T}\times\mathbf{N}\to\mathbf{N}$ be computable. Define*

$$\mathbf{O}(T)=\left\{F\mid\exists c\in\mathbf{N}\ \ [F\leq_2^* cT]\right\}.$$

It is not clear if there is a reasonably weak condition on T so that we can have a computable F and $\mathbf{C}(F)=\mathbf{O}(T)$. A positive result to this concern requires a Union Theorem.

At type-1, it is easy to show that the *finite invariant* closure of a complexity class is *recursively enumerable* [4]. However, not every complexity class itself can be recursively enumerated. When the resource bound t is very small (namely, very *dishonest*), the complexity class determined by t is unlikely to be recursively enumerable [4, 17]. On the other

 © Chung-Chih Li

hand, if t is big enough to bound all *finite support* functions[6] almost everywhere, then the complexity class determined by t is recursively enumerable. In particular, if t is nontrivial, i.e., $t(x) \geq |x| + 1$ for all $x \in \mathbf{N}$, then all finite support functions are contained in the complexity class determined by t (see [5], Section 9.4). The intuitive reason behind this is that, if the bound t allows to compute every finite support function almost everywhere, then we can patch a program at finitely many places with cost bounded by t almost everywhere. In such a way, we can exactly enumerate the complexity class determined by t. At type-2, we have the same situation. To recursively enumerate $\mathbf{C}(T)$, we need a notion of non-triviality for T.

Definition 18 *Let $T : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be computable. T is said to be nontrivial if and only if there is a constant $c \in \mathbf{N}$ such that, for every minimal locking fragment (τ, x) of T, we have that, if $(f, x) \in (\!(\tau, x)\!)$ and $\sigma \subseteq \tau$, then $T(f, x) \geq c(|\sigma| + |x|)$.*

Note that, the constant c in Definition 18 depends on specific conventions on OTM. Although we may not be interested in finding out what c really is, we can't drop this constant until a linear speedup theorem is formally proven. Intuitively, a nontrivial computable resource bound T must allow any OTM to check whether or not (f, x) is in $(\!(\tau, x)\!)$ under the cost bounded by T as long as (τ, x) is a fixed minimal locking fragment of T. This property serves the same purpose of non-triviality of classical type-1 resource bounds. We obtain the following theorem.

Theorem 19 *Let $T : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be computable and nontrivial. Then, the complexity class $\mathbf{C}(T)$ is recursively representable.*

Proof: We first fix some notations. Recall that every $\sigma \in \mathcal{F}$ is represented by a unique canonical index. Let $\sigma^{\sim 0} \in \mathcal{T}$ denote the zero extension of $\sigma \in \mathcal{F}$. That is, $\sigma^{\sim 0}(x) = \sigma(x)$ when $x \in \mathsf{dom}(\sigma)$; $\sigma^{\sim 0}(x) = 0$ otherwise. Let $\langle \cdot, \cdot \rangle : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ be a standard pairing function. Together with the canonical indexing of $\mathcal{F}$, we have $\langle \sigma, x \rangle \in \mathbf{N}$ for every $\sigma \in \mathcal{F}$ and $x \in \mathbf{N}$.

Let $T : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be computable and nontrivial. Unlike the proof for its type-1 counterpart, we are not going to enumerate all finite invariants of $\mathbf{C}(T)$.[7] Instead, we directly argue that there is a recursive function g such that,

$$\mathbf{C}(T) = \{\widehat{\varphi}_{g(e,a,b)} \mid e, a, b, \in \mathbf{N}\}.$$

[6] A function f is finite support if the value of f is 0 almost everywhere.

[7] Unlike the proof of the theorem's type-1 counterpart, we are not going to enumerate all finite invariants of $\mathbf{C}(T)$. In fact, we haven't had a precise definition of finite invariant for type-2 functionals.

© Chung-Chih Li

With a proper modified *S-m-n theorem* for OTMs on type-0 arguments,[8] we can construct a recursive $g : \mathbf{N} \times \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ such that, for every $e, a, b \in \mathbf{N}$ and $(f, x) \in \mathcal{T} \times \mathbf{N}$,

$$\widehat{\varphi}_{g(e,a,b)}(f,x) = \begin{cases} \widehat{\varphi}_e(\tau^{\sim 0}, x) & \text{if } \forall \langle \sigma, y\rangle \leq a\ [\widehat{\Phi}_e(\sigma^{\sim 0}, y) \leq T(\sigma^{\sim 0}, y) + b] \text{ and} \\ & \quad \forall \langle \sigma, y\rangle \leq \langle \tau, x\rangle [\forall \eta \subseteq \sigma (a < \langle \eta, y\rangle) \Rightarrow \widehat{\Phi}_e(\sigma^{\sim 0}, y) \leq T(\sigma^{\sim 0}, y)], \\ & \text{where } (\tau, x) \text{ is a locking fragment of } T \text{ on } (f, x). \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

If $F \in \mathbf{C}(T)$, then there is a $\widehat{\varphi}$-program e for F such that, $X_{[\widehat{\Phi}_e > T]}$ is compact in $\mathbb{T}(\widehat{\Phi}_e)$. Consider the condition clause of the if statement in (8). We observe that a serves as the upper bound of the basic open set in the finite cover of $X_{[\widehat{\Phi}_e > T]}$, and b serves as the extra cost for the computation of $\widehat{\varphi}_e$ on $X_{[\widehat{\Phi}_e > T]}$ (which is bounded, by a direct application of Theorem 14). Thus, if we choose sufficiently large a and b, then the condition of the if statement in (8) will always be true, and hence we will have $\widehat{\varphi}_{g(e,a,b)} = \widehat{\varphi}_e$, i.e., $F = \widehat{\varphi}_{g(e,a,b)}$. Note that, if $\widehat{\Phi}_e \leq_2^* T$, all queries made outside a locking fragment of T during the course of $\widehat{\varphi}_e$'s computation are not necessary. Otherwise, $\widehat{\Phi}_e \not\leq_2^* T$ because the answer returned from the oracle can be arbitrarily large.[9] Thus, if the condition in (8) holds, then $\widehat{\varphi}_e(f, x) = \widehat{\varphi}_e(\tau^{\sim 0}, x)$, although $\widehat{\varphi}$-program e may query outside $\mathsf{dom}(\tau)$.

Next, we argue that, for every $e, a, b \in \mathbf{N}$, we have $\widehat{\varphi}_{g(e,a,b)} \in \mathbf{C}(T)$. Note that, we do not argue that $\widehat{\Phi}_{g(e,a,b)} \leq_2^* T$, but that there is a $\widehat{\varphi}$-program e' such that, $\widehat{\varphi}_{g(e,a,b)} = \widehat{\varphi}_{e'}$ and $\widehat{\Phi}_{e'} \leq_2^* T$. In other words, there is a patched version of $\widehat{\varphi}$-program $g(e, a, b)$ with complexity asymptotically bounded by T. Fix $e, a, b \in \mathbf{N}$. Let $F = \widehat{\varphi}_{g(e,a,b)}$ and $S \subseteq \mathcal{T} \times \mathbf{N}$ be the set on which the $\widehat{\varphi}$-program $g(e, a, b)$ uses $\widehat{\varphi}_e(\tau^{\sim 0}, x)$ as its output, i.e., for every $(f, x) \in S$, the condition of the if statement in (8) holds. We have two cases: (i) S is compact in $\mathbb{T}(F)$ and (ii) S is not compact in $\mathbb{T}(F)$.

Case (i)**:** S is compact in $\mathbb{T}(F)$. In this case, we fix a finite open cover for S as follows:

$$\mathcal{O} = \{(\!(\sigma_0, x_0)\!), (\!(\sigma_1, x_1)\!), \ldots, (\!(\sigma_n, x_n)\!)\}. \tag{9}$$

That is, for each $(\!(\sigma_i, x_i)\!) \in \mathcal{O}$, (σ_i, x_i) is a minimal locking fragment of F. Let $t \in \mathcal{F}$ be a finite function such that, $\mathsf{dom}(t) = \{0, 1, \ldots, n\}$ and for each $i \leq n$, $t(i) = \widehat{\varphi}_e(\sigma_i^{\sim 0}, x_i)$. Then, we construct a $\widehat{\varphi}$-program e' as follows:

$$\widehat{\varphi}_{e'}(f,x) = \begin{cases} t(i) & \text{if } \sigma_i \subset f \text{ and } x_i = x \text{ for some } (\!(\sigma_i, x_i)\!) \in \mathcal{O}; \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

Clearly, $\widehat{\varphi}_{e'} = F$. Since T is nontrivial and every (σ_i, x_i) is either a minimal locking fragment of T or σ_i is a proper subset of τ where (τ, x_i) is a minimal locking fragment of T, it follows that, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, we can check the condition in (10) with cost bounded by T. Therefore, $F \in \mathbf{C}(T)$ in this case.

[8] Obviously, we do not have a *S-m-n theorem* for OTMs on type-1 arguments.

[9] Here we assume whatever returned from the oracle has to be scanned at least once. We call the model of OTM's under this convention *Answer-Length-Model* in [18].

© Chung-Chih Li

Case (ii)**:** S is not compact in $\mathbb{T}(F)$. In this case, we fix an open cover for S:

$$\mathcal{O} = \{(\!(\sigma_0, x_0)\!), (\!(\sigma_1, x_1)\!), \ldots\}$$

Similarly, for each $(\!(\sigma_i, x_i)\!) \in \mathcal{O}$, (σ_i, x_i) is a minimal locking fragment of F. Let (τ, x) be some minimal locking fragment of T. Consider the second clause of the condition in (8):

$$\forall \langle \sigma, y\rangle \leq \langle \tau, x\rangle [\forall \eta \subseteq \sigma (a < \langle \eta, y\rangle) \Rightarrow \widehat{\Phi}_e(\sigma^{\sim 0}, y) \leq T(\sigma^{\sim 0}, y)] \tag{11}$$

By the assumption of this case, there are infinitely many (τ, x) such that (11) is true. It follows that, (11) is true on all but finitely many (τ, x). Thus, $X_{[\widehat{\Phi}_e > T]}$ is compact in $\mathbb{T}(T)$. On the other hand, there are only finitely many (τ, x) such that, on $(f, x) \in (\!(\tau, x)\!)$, the $\widehat{\varphi}$-program $g(e, a, b)$ outputs 0 instead of $\widehat{\varphi}_e(f, x)$. Thus, the set

$$S' = \left\{(f, x) \;\middle|\; F(f, x) \neq \widehat{\varphi}_e(f, x)\right\}$$

is also compact in $\mathbb{T}(T)$. Thus, we can patch e on S' with computational cost bounded by T. Therefore, $F \in \mathbf{C}(T)$. □

5.2 Type-2 Complexity Theorems – Rabin's, Recursive Relatedness, and Gap Theorems

Here we demonstrate the type-2 analog of three interesting complexity theorems in classical complexity theory: Rabin's theorem [24], recursive relatedness theorem [2], and Gap theorem [3]. Our purpose is to show that our notion of type-2 asymptotic approach is a reasonable one that can lead to a full scale investigation of type-2 complexity theory.

Arbitrarily Complex At Type-1, there is no time-bound that is big enough to determine a complexity class including all recursive function. This claim is rather trivial if the output of the recursive functions can be arbitrarily big. To make it more interesting, in [24] Rabin refines the claim by restricting the recursive functions to highly dishonest ones (0-1 valued). Rabin's theorem states that, for any recursive function t, there is a 0-1 valued recursive function f such that $f \notin DTIME(t)$. The theorem is proven by a technique known as *cancellation argument*, which combines the diagonalization method and the priority argument without *injury*[10]. We modify Rabin's proof and obtain an analogous type-2 result as follows.

Theorem 20 (Type-2 Rabin's Theorem) *For any computable $T : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, there is a 0-1 valued computable $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that $F \notin \mathbf{C}(T)$.*

Proof: Fix any computable $T : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$. We construct an algorithm for $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ shown in Figure 1.

[10] See [23], page 29, for elaboration.

© Chung-Chih Li

```
Program F(f : 𝒯; x : N)
    Q ⟵ ∅;
    for w = 0 to x do
        S_w ⟵ { k | k ≤ w, k ∉ Q, Φ̂_k(f, w) ≤ T(f, w) };
        if S_w ≠ ∅ then Q ⟵ Q ∪ {min(S_w)};
    end for;
    if S_x = ∅ then return 0;
    e ⟵ min(S_x);                                   /* cancel e */
    if Φ̂_e(f, x) > T(f, x) then return 0;
    return (1 + φ̂_e(f, x)  mod 2);
End program
```

Figure 1: A Type-2 Cancellation Algorithm for Rabin's Theorem

Note that, on every $(f,x) \in \mathcal{T} \times \mathbf{N}$, the Q in the algorithm is finite. Since T is computable and total, it follows that, for every $(f,w) \in \mathcal{T} \times \mathbf{N}$, $\widehat{\Phi}_k(f,w) \leq T(f,w)$ is effectively decidable and hence S_w in the loop of the algorithm can be obtained effectively. In other words, the loop always terminates. Moreover, $\widehat{\Phi}_e(f,x) > T(f,x)$ is also effectively decidable. Thus, F is a total computable functional. Clearly, F is 0-1 valued.

Fix an e such that $\widehat{\varphi}_e \in \mathbf{C}(T)$, i.e, $X_{[\widehat{\Phi}_e > T]}$ is compact in $\mathbb{T}(\widehat{\varphi}_e)$. We shall show that the $\widehat{\varphi}$-program e does not compute F. Since $X_{[\widehat{\Phi}_e > T]}$ is $\mathbb{T}(\widehat{\varphi}_e)$-compact, there is an $a \geq e$ such that,

$$(\forall x \geq a)(\forall f \in \mathcal{T})\ [\widehat{\Phi}_e(f,x) \leq T(f,x)]. \tag{12}$$

Fix an $f \in \mathcal{T}$. Suppose that x_0 is the least number such that, $e \leq x_0$ and $\widehat{\Phi}_e(f,x_0) \leq T(f,x_0)$. Consider the computation of F on (f,x_0). According to the algorithm, e will be in S_{x_0} after the loop is done. If e is the least number in S_{x_0}, then $F(f,x_0) = (1 + \widehat{\varphi}_e(f,x_0) \mod 2)$. If e is not the least number in S_{x_0}, we shall prove that sooner or later there is some $x \geq x_0$ such that, $e = \mathsf{min}(S_x)$. Clearly, for such x, we have $\widehat{\Phi}_e(f,x) \leq T(f,x)$.

Suppose that $k = \mathsf{min}(S_{x_0})$ and $k \neq e$. The for loop assures that every index will be cancelled at most once on each f. Thus, for any $x > x_0$, $k \notin S_x$. To see this, consider the execution of the loop: when $w = x_0$, $k = \mathsf{min}(S_w)$; hence $k \in Q$ and k will not be selected again for S_x. Since there are only finitely many indices less than e, and each index is cancelled at most once, it follows that if x is sufficiently large, then $e \leq \mathsf{min}(S_x)$. Also, because of (12), if x is sufficiently large, then $\widehat{\Phi}_e(f,x) \leq T(f,x)$. This means that $e \in S_x$ unless e has been cancelled for some $w < e$, i.e., $e = \mathsf{min}(S_w)$ and hence $e \in Q$. Thus, there is an x such that $e = \mathsf{min}(S_x)$ and $F(f,x) = (1 + \widehat{\varphi}_e(f,x) \mod 2)$. Therefore, $F \neq \widehat{\varphi}_e$. □

Recursive Relatedness Theorem As we mentioned earlier, recursive relatedness is a bridge for complexity theorems between different complexity measures. A type-2 analog

© Chung-Chih Li

will be also essential if we want to further abstract away from any specific model of type-2 computation. We thus formulate a type-2 Recursive Relatedness Theorem in the following.

Theorem 21 (Type-2 Recursive Relatedness Theorem) *Let $\langle\Phi\rangle_{i\in\mathbf{N}}$ and $\langle\Psi\rangle_{i\in\mathbf{N}}$ be two complexity measures for OTM's. (I.e., $\langle\Phi\rangle_{i\in\mathbf{N}}$ and $\langle\Psi\rangle_{i\in\mathbf{N}}$ satisfies Blum's two axioms [2].) Then, there is a computable functional $R : \mathcal{T}\times\mathbf{N}\times\mathbf{N}\to\mathbf{N}$ such that, for every $i\in\mathbf{N}$, we have*

1. $\Phi_i \leq_2^* \lambda F, x\cdot R(F, x, \Psi_i(F, x))$, *and*
2. $\Psi_i \leq_2^* \lambda F, x\cdot R(F, x, \Phi_i(F, x))$. □

We omit the proof for it can be obtained from the original proof with some minor modification. It follows from this theorem that, any result we prove about a particular complexity measure, there is a "recursively related" result about each other complexity measure.

Gap Theorem The operation of an effective operator of type $\mathcal{R}\times\mathbf{N}\to\mathbf{N}$ is indeed a special case of type-2 computations where the type-1 input is restricted to $\mathcal{R}$ (recursive functions). However, the Operator Gap Theorem [6, 32] does not imply that we can directly obtain a gap theorem at type-2. In fact, we prove that if we allow the gap factor to be a type-2 computable functional (not just an operator), we can uniformly construct a type-2 computable functional that can inflate every type-2 complexity class [18]. For simplicity, here we restrict the gap factor to recursive functions. We obtain the following result.

Theorem 22 (Type-2 Gap Theorem) *Given any strictly increasing recursive function $g : \mathbf{N}\to\mathbf{N}$, there is a computable functional T such that, $\mathbf{C}(T) = \mathbf{C}(g\circ T)$.*

Sketch of Proof: To proof this theorem, we first accept a convention that the OTM has to scan (read) every bit of the oracle answer at least once; otherwise an opposite theorem can be proven [18]. In other words, the cost of a query is at least the length of the answer. This model is called *Answer-Length-Cost Model* [26]. We use the predicate $P(f, x, k)$ defined as follows to determine the value of $T(f, x)$.

$$P(f,x,k) \equiv \begin{array}{l}\text{Every computation of } \widehat{\varphi}_1, \widehat{\varphi}_2, \ldots, \widehat{\varphi}_x \text{ on } (f,x) \text{ is either: } (i) \text{ making a oracle query outside } \{0,1,\ldots,x\},\\ \text{or } (ii) \text{ halts in } k \text{ steps or does not halt in } g(k) \text{ steps.}\end{array}$$

It is clear that the predicate above is computable. The the predicate, (ii) essentially comes from the idea of the original proof. For (i), we observe that, under our convention, if $T(f, x)$ converges on a segment $\sigma\subset f$ with $\sigma\subseteq\{0, 1, \ldots, x\}$, then so does $g(T(f, x))$ and no OTM e that queries beyond $\mathsf{dom}(\sigma)$ can have $\widehat{\Phi}_e \leq_2^* g\circ T$.

 © Chung-Chih Li

6 Conclusion

As a matter of fact, general type-2 complexity theory is still an unclear territory. Many applications of type-2 (or higher) computations such as machine learning and programming language use their own approaches to address their own complexity issues. It is usually difficult to apply one approach that has been successful in one application to another. We believe that a workable notion of asymptotic behaviors of type-2 algorithms is the first step in the search of a standard framework for the study of type-2 complexity. And we hope that our notion of $\leq_2^*$ is a such step.

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] A. Borodin. Complexity classes of recursive functions and the existence of complexity gaps. *Conference Record of ACM Symposium on the Theory of Computing*, pages 67–78, 1969.

[4] A. Borodin. Computational complexity and the existence of complexity gaps. *Journal of the ACM*, 19(1):158–174, 1972.

[5] Walter S. Brainerd and Landweber Lawrance H. *Theory of Computation*. John Wiley & Sons, New York, 1974.

[6] Robert L. Constable. The operator gap. *Journal of the ACM*, 19:175–183, 1972.

[7] Robert L. Constable. Type two computational complexity. In *Proceedings of the 5^{th} ACM Symposium on the Theory of Computing*, pages 108–122, 1973.

[8] Stephen Cook and Alasdair Urquhart. Functional interpretation of feasibly constructive arithmetic. *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing*, pages 107–112, 1989.

[9] Stephen A. Cook and Bruce M. Kapron. Characterization of the basic feasible functions of finite type. *Proceedings of the 30th Annual IEEE Symposium on the Foundations of Computer Science*, pages 154–159, 1989.

[10] Martin Davis. *Computability and Unsolvability*. McGraw-Hill, 1958. First reprinted by Dover in 1982.

 © Chung-Chih Li

[11] R.O. Gandy and J.M.E. Hyland. Computable and recursively countable functions of higher type. *Logic Colloquium*, 76:405–438, 1977.

[12] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transitions of the American Mathematics Society*, pages 285–306, May 1965.

[13] Bruce M. Kapron and Stephen A. Cook. A new characterization of Mehlhorn's polynomial time functionals. *Proceedings of the 32th Annual IEEE Symposium on the Foundations of Computer Science*, pages 342–347, 1991.

[14] Bruce M. Kapron and Stephen A. Cook. A new characterization of type 2 feasibility. *SIAM Journal on Computing*, 25:117–132, 1996.

[15] Steve C. Kleene. Turing-machine computable functionals of finite types II. *Proceedings of London Mathematical Society*, 12:245–258, 1962.

[16] Steve C. Kleene. Recursive functionals and quantifies of finite types II. *Transitions of the American Mathematics Society*, 108:106–142, 1963.

[17] L.H. Landweber and E.R. Robertson. Recursive properties of abstract complexity classes. *ACM Symposium on the Theory of Complexity*, May 1970.

[18] Chung-Chih Li. Type-2 complexity theory. Ph.d. dissertation, Syracuse University, New York, 2001.

[19] Chung-Chih Li and James S. Royer. On type-2 complexity classes: Preliminary report. *Proceedings of the Third International Workshop on Implicit Computational Complexity*, pages 123–138, May 2001.

[20] E. McCreight and A. R. Meyer. Classes of computable functions defined by bounds on computation. *Proceedings of the First ACM Symposium on the Theory of Computing*, pages 79–88, 1969.

[21] A. Nerode. General topology and partial recursive functionals. *Talks Cornell Summ. Inst. Symb. Log., Cornell*, pages 247–251, 1957.

[22] Piergiorgio Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science Publishing, North-Holland, Amsterdam, 1989.

[23] Piergiorgio Odifreddi. *Classical Recursion Theory, Volume II*, volume 143 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science Publishing, North-Holland, Amsterdam, 1999.

[24] M.O. Rabin. Degree of difficulty of computing a function and a partial ordering of recursive sets. Technical Report 2, Hebrew University, 1960.

 © Chung-Chih Li

[25] Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967. First paperback edition published by MIT Press in 1987.

[26] 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.

[27] Joel I. Seiferas and Albert R. Meyer. Characterization of realizable space complexities. *Annals of Pure and Applied Logic*, 73:171–190, 1995.

[28] Anil Seth. Complexity theory of higher type functionals. Ph.d. dissertation, University of Bombay, 1994.

[29] Robert I. Soare. *Recursively Enumerable Sets and Degrees*. Perspectives in Mathematical Logic. Springer-Verlag, New York, 1987.

[30] D.M. Symes. The extension of machine independent computational complexity theory to oracle machine computation and the computation of finite functions. Ph.d. dissertation, University of Waterloo, Oct. 1971.

[31] V.A. Uspenskii. On countable operations (Russian). *Doklady Akademii Nauk SSSR*, 103:773–776, 1955.

[32] Paul Young. Easy construction in complexity theory: Gap and speed-up theorems. *Proceedings of the American Mathematical Society*, 37(2):555–563, February 1973.

© Chung-Chih Li