ASYMPTOTIC BEHAVIORS OF TYPE-2 ALGORITHMS AND INDUCED BAIRE TOPOLOGIES*

Chung-Chih Li
Computer Science Department
Lamar University
Beaumont, Texas, USA
licc@hal.lamar.edu

Abstract We 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" for type-2 complexity theory. A natural reference for type-2 computations is the standard Baire topology. However, we point out some serious drawbacks of this and introduce an alternative topology for describing compact sets. Following our notion explicit type-2 complexity classes can be defined in terms of resource bounds. We show that such complexity classes are recursively representable; namely, every complexity class has a programming system. We also prove type-2 analogs of Rabin's Theorem, Recursive Relatedness Theorem, and Gap Theorem to provide evidence that our notion of type-2 asymptotic is workable. We speculate that our investigation 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 Complexity, Type-2 Asymptotic Notation, Baire Topology.

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,

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

*A full version with detailed proofs of the theorems in this paper is available at
http://hal.lamar.edu/~licc/T2Asy/Full_T2AsyTCS2004.pdf

Based on the notion above, Hartmanis and Stearns [8] 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* [18] 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 (Θ, Ω, O) in algorithm analysis. The most important consequence of using asymptotic notations, in our opinion, is not only 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, 24], the Union Theorem [13], the Gap Theorem [3, 5, 24], the Compression Theorem [2], the Honesty Theorem [13], and so on (see [20] for more), are all proven by a mathematical technique called *priority method*. The method argues that the required properties (behaviors) of a required program will be fulfilled eventually in the process of its construction. In other words, we allow some finitely many violations. Likewise, the *recursive relatedness theorem* [2] is also proven based on the notion of asymptotic behaviors, which is the foundation for lifting different complexity measures into a certain degree of abstraction such as the two machine independent axioms proposed by Blum [2]. It is worth to note that the asymptotic notation is not arbitrary. Instead, it is justified by a fact that any program can be patched on some finitely many inputs by using a finite table to avoid expensive computations on those particular inputs. Thus, theoretically, we can use $\leq$ and $\leq^*$ in (2) alternatively without changing the underlying complexity structure of computable functions[1].

When we shift our attention to higher ordered computation (in particular, type-2 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. A primary 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 [7] 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 convention, we consider type-0 $\subset$ type-1 $\subset$ type-2. We are only interested

[1] In fact, this is an overstatement. Here we overlook the honesty property of t, which is a necessary condition for the statement. Nevertheless, the honesty condition is rather weak for most reasonable resource bounds.

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: $F(f, x) = f(x)$; $G(f, x) = f(f(x))$; $H(f, x) = \sum_{i=0}^{x} f(i)$; $\Gamma(f) = f \circ f$, where $\circ$ is 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 [23] and Nerode [14]: *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 $\boldsymbol{compact}$ *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 $\boldsymbol{monotone}$ *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 are the key properties of computable functionals in defining our topologies for the concerned computation. 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 [18]. As for classical Turing Machines, we can fix a programming system $\langle \widehat{\varphi} \rangle_{i \in \mathbf{N}}$ associated with a complexity measure $\langle \widehat{\Phi} \rangle_{i \in \mathbf{N}}$ (say, the number of steps performed) for OTM's. 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 part of the current type-2 complexity theory.

2. An Outlook of Present Complexity Theory at Type-2

Cook and Kapron defined *second-order polynomials* [9] in order to characterize the set of type-2 Basic Feasible Functionals (BFF here after) [6]. Their framework

requires a rather artificial function called the *length function*, which is served as the type-2 analog of $|x|$, where $|x|$ is the length of the bit string representing $x \in \mathbf{N}$. For $f \in \mathcal{T}$, the length function of f is defined by

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

Thus, $|f|(n)$ is the maximum length of the values of f on input with length $\leq n$. BFF, to some degree, is seen as the type-2 analog of **P**. Inside BFF, how conceivable is the use of second-order polynomials together with the length functions? We present an easy example to show that some results may drift away from our intuition. Consider

$$\begin{aligned} F(f,x) = \ & 2^{\mathsf{max}\{f(2^{|x|}),f(2^{|x|}-1),\ldots,f(2^{|x|}-(|x|-1))\}}; \\ G(f,x) = \ & \begin{cases} 2^{2^{1000}} & \text{if } x = 0 \text{ and } f(2^{|x|}) = 0; \\ 2^{\mathsf{min}\{f(2^{|x|}),f(2^{|x|}-1),\ldots,f(2^{|x|}-(|x|-1))\}} & \text{otherwise.} \end{cases} \end{aligned} \tag{3}$$

Let $\widehat{\Phi}_F$ and $\widehat{\Phi}_G$ denote their cost functions (e.g., numbers of steps performed). We observe that the major cost of computing the two functionals is querying the oracle. For each query q, at least $O(|q| + |f(q)|)$ steps are needed (for placing the query and reading the answer). Thus, the max and min functions above need $|x|$ many queries and each query need $O(|x| + |f(2^{|x|})|)$ steps. In terms of length functions, both functionals are bounded by $O(|x| \times (|x| + |f|(|x|))$, and hence by a second-order polynomial $\boldsymbol{p}$ defined as $\boldsymbol{p}(\ell, x) = c(x^2 + x \cdot \ell(x))$, where $c \in \mathbf{N}$. However, we also observe that, unless $x = f(2^{|x|}) = 0$, we have $\widehat{\Phi}_G(f,x) \leq \widehat{\Phi}_F(f,x)$. In other words, in *most cases* we have $\widehat{\Phi}_G(f,x) \leq \widehat{\Phi}_F(f,x)$, but we have difficulty to describe this situation in terms of second-order polynomials and length functions. 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" "affordable" exceptional cases? On what ground we can justify our intuition that G is easier than F?

For a general type-2 complexity theory to begin with, arguably, 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.[2] 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 [21] 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 some classical complexity results such as the Gap theorem and the Union theorem may be proven. 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

[2] 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.

of the two theorems 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 inputs (because some of them 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. Based on this observation, in early 70's Symes [22] presented an axiomatic approach for type-2 complexity theory. The axiomatic system was modified from Blum's. Symes required a computation (represented by a computation tree) of the concerned type-2 functionals to be explicitly provided as an input. The machine will be shut down if the provided computation is not consistent with the "actual" computation of the machine. In his proofs, $\leq^*$ was used as the context: "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 for practice. No one can provide a computation tree before the computation begins. All we can do is to enumerate all possible computations only for theoretical investigation.

In the following section we introduce a new idea to define a workable type-2 almost-everywhere relation, $\leq_2^*$. As "compact" used in topology to some extent is considered as a surrogate for "finite" and "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:. Let $\mathbb{N}$ be the discrete topology on **N**. The space $\mathcal{T}$ is called Baire space. The Baire topology [1, 15, 18] is denoted by $\mathbb{T}$, in which a basic open set is the set of all total extensions of some finite function. Let $\mathbb{T} \times \mathbb{N}$ denote the product topology of $\mathbb{T}$ and $\mathbb{N}$. 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 $\{(f, x) | F(f, x) \leq G(f, x)\}$. Similarly, $X_{[F=X]}, X_{[F>G]}$ will be used in the same way. A type-2 functional F is said to be computable if there is an OTM with index e that computes F (i.e., $F = \widehat{\varphi}_e$). For $F, G : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$, a straightforward analog of $\leq^*$ would be:

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

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 finitely many computations of F and G, the result of F is less than or equal to the result of G.*" This in fact is Symes' idea. We put the subscript "$_2$" in $\leq_2^*$ to reflect the type of its operands. We will use "computation" to mean "terminating computation" for the time being. 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 achieve.

Goal 1: 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 that we can patch a program, and the second one assures 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 at type-2. Unfortunately, the two properties conflict; they hurt each other. In the end, we give up transitivity, which does not seem too essential to our primary purpose: a workable notion of type-2 asymptotic behaviors for proving theorems. The following standard theorem hints a possible way to formalize "$\leq_2^*$".

THEOREM 2 *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 $(f, x) \in (\mathcal{T} \times \mathbf{N} - X)$ we have $F(f, x) \leq G(f, x)$. We restate this in the following definition.

DEFINITION 3 *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 $\mathbb{T} \times \mathbb{N}$.*

We change the superscript to "$+$" to reflect the conclusion we will discuss in a moment that this definition is too strong for our purposes. Nevertheless, the following theorem shows that relation $\leq_2^+$ meets our Goal 2.

THEOREM 4 *Relation $\leq_2^+$ is transitive over type-2 continuous functionals.* □

A standard property of Baire topology says that, 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 5 *If F and G are continuous and $F \leq_2^+ G$, then there exists $c \in \mathbf{N}$ such that, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, $F(f, x) \leq G(f, x) + c$.* □

Thus, if $F \leq_2^+ G$, then adding some constant value c to G allows us to bound F everywhere. If we consider G as some sort of resource bound, we speculate that a constant or *linear speedup theorem* may be proven. In other words, we can patch the program for F to remove the extra constant cost c. Comparing to (1), the similarity between $f \leq^* g$ and $F \leq_2^+ G$ can be easily seen:

$$F \leq_2^+ G \iff \exists f_0 \in \mathcal{T} \exists x_0 \in \mathbf{N} \forall f \in \mathcal{T} \forall x \in \mathbf{N} \\ [(f > f_0) \vee (x > x_0) \Rightarrow F(f, x) \leq G(f, x)].$$

This suggests that Definition 3 might be a right choice. However, $\leq_2^+$ has a fatal problem that discourages us to move any further. We show that $\leq_2^+$ in fact is an empty

notion and any possible definition for type-2 asymptotic behaviors based on $\leq_2^+$ will not give any flexibility.

THEOREM 6 *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* [15]. 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 need a topology that can provide enough compact sets for describing "small" sets. In other words, a coarser topology is needed.

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 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. In the end, we give up transitivity in order to have a workable notion of type-2 asymptotic behaviors. Let $F(f, x) \downarrow = y$ denote the case that F is defined on (f, x) and its value is y.

DEFINITION 7 *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*

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

*If (σ, x) is a locking fragment of F and, for every $\tau \subset \sigma$, (τ, x) is not a locking fragment of F, then (σ, x) is said to be a **minimal 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, if (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 $x_1, x_2 \in \mathbf{N}$,

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

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 $a \in \mathbf{N}$, if $f \neq g$, then there exist σ, τ, and k such that, $\sigma \subset f, \tau \subset g$, $k \in \mathsf{dom}(\sigma) \cap \mathsf{dom}(\tau)$, and $\sigma(k) \neq \tau(k)$. 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 participated functionals.

DEFINITION 8 *Given a finite number of continuous functionals, $F_1, \ldots, F_n$, let $\mathbb{T}(F_1, \ldots, F_n)$ denote the topology determined by $F_1, \ldots, F_n$ as follows. For each $(f, a) \in \mathcal{T} \times \mathbf{N}$, let (σ_i, a) be the minimal locking fragment of F_i on (f, a). Take $((\sigma, a)) = ((\sigma_1, a)) \cap ((\sigma_2, a)) \cap \cdots \cap ((\sigma_n, a))$ as a basic open set of $\mathbb{T}(F_1, \ldots, F_n)$.*

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 to each of $F_1, F_2, \ldots,$ and F_n. However, given any two functionals, F_1 and F_2, the topologies $\mathbb{T}(F_1)$ and $\mathbb{T}(F_2)$ are determined by different basic open sets, and hence the two topologies do not share the same set of compact sets. This in fact is the inherited difficulty of having a transitive relation.

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

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

DEFINITION 9 *Let $F_1, F_2 : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be continuous. Define*

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

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 10 *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 $\mathbb{T}(F_2)$. □

Thus, Goal 1 fails,[3] but the statement of Goal 1 may be too strong in the context of type-2 computation if the real purpose behind is to patch programs. We have the following theorem to support our definition.

[3] We could have defined Definition 9 as $F_1 \leq_2^\star F_2$ if and only if $X_{[F_1 \leq F_2]}$ is co-compact in $\mathbb{T}(F_1, F_2)$. In such a way we will have a finer topology so that every involved functional is also continuous in $\mathbb{T}(F_1, F_2)$. It follows that we can have a result opposite to Theorem 10. 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 $X_{[F_1 > F_2]}$.

THEOREM 11 *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 $X_{[F_1 > F_2]}$.* □

We omit the proof, which is obvious from the definition of $\leq_2^*$. If F_1 in the above theorem is the cost function of some OTM, we can patch the machine so that the complexity of the patched machine is bounded by F_2 everywhere.

THEOREM 12 *The relation $\leq_2^*$ is not transitive.*

Proof: The idea is that the relations $F_1 \leq_2^* F_2$ and $F_2 \leq_2^* F_3$ hold based on two unrelated topologies $\mathbb{T}(F_1)$ and $\mathbb{T}(F_2)$, respectively. Thus, $X_{[F_2 > F_3]}$ may not be compact in $\mathbb{T}(F_1)$. Consequently, the set $X_{[F_1 > F_3]}$, which is a subset of $X_{[F_1 > F_2]} \cup X_{[F_2 > F_3]}$ may not be compact in $\mathbb{T}(F_1)$. Consider the following example.

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

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

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

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

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

Since the only basic open set of $\mathbb{T}(F_2)$ that contains $X_{[F_2 > F_3]}$ is $(\!(\sigma, 0)\!)$ with $\sigma(0) = 0$ and $\mathsf{dom}(\sigma) = \{0\}$, it follow that $X_{[F_2 > F_3]}$ is compact in $\mathbb{T}(F_2)$, and hence $F_2 \leq_2^* F_3$. We observe F_1 and F_3 to have

$$X_{[F_1 > F_3]} = \{(f,0) \mid f(0) = 0 \text{ and } f(1) = 3 + 6k \text{ with } k \in \mathbf{N}\}.$$

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

$$\mathcal{O} = \{(\!(\sigma_n, 0)\!) \mid n = 3 + 6k \text{ with } k \in \mathbf{N}\}.$$

Clearly, $\mathcal{O}$ is an open cover for $X_{[F_1 > F_3]}$ without finite subcover. Thus, $X_{[F_1 > F_3]}$ 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). We simply compare $\widehat{\Phi}_F$ and $\widehat{\Phi}_G$. Assume $|0| = 1$ under some coding convention. Thus, $2^{|x|} = 2$ if $x = 0$. 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

$\mathbb{T}(\widehat{\Phi}_G)$[4], 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 our intuitive understanding about the complexity of G and F.

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

5.1 Type-2 Complexity Classes

In [12, 11] 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 .

$$F \in \mathbf{C}(\beta) \Longrightarrow \exists e \left[\widehat{\varphi}_e = F \ \wedge \ \widehat{\Phi}_e \leq_2^* F_\beta \right]. \tag{6}$$

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 (6) 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 13 *Let $T : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be computable. Define*

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

Here we point out a fact with detailed explanation omitted that the notions of $\mathbf{C}(T)$ and $\mathbf{C}(\beta)$ are not equivalent. Nevertheless, we speculate that the type-2 almost everywhere relation involved in Definition 13 will let us directly modify the proofs given in [11] for type-2 Speedup Theorem, Gap Theorem, Union Theorem, Compression Theorem, and so on.

An analog big-O notation for type-2 algorithms can be directly given as follows:

DEFINITION 14 *Let $T : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be computable. Define*

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

[4]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 a machine for G (some unnecessary queries may be 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.

We do not know yet if there is a computable functional F such that $\mathbf{C}(F) = \mathbf{O}(T)$ in general. A positive result to this question requires a Union Theorem.

At type-1, it is easy to show that the *finite invariant* closure of a complexity class is *recursively enumerable* [3]. 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 [3, 10]. On the other hand, if t is big enough to bound all *finite support* functions[5] 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 [4], 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 15 *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 15 depends on OTM's conventions. 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 allows an OTM to check whether or not (f, x) is in $(\!(\tau, x)\!)$ at computational 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 16 *Let $T : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be computable and nontrivial. Then, the complexity class $\mathbf{C}(T)$ is recursively enumerable.*

Sketch of Proof: 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 $\mathbf{C}(T)$.[6] Instead, we directly argue that there is a recursive function g such that,

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

With a proper *S-m-n theorem* on type-0 arguments,[7] 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 $(f, x) \in \mathcal{T} \times \mathbf{N}$,

[5] A function f is finite support if the value of f is 0 almost everywhere.
[6] In fact, we haven't had a precise definition of finite invariant for type-2 functionals.
[7] Obviously, we do not have an *S-m-n theorem* for OTMs on type-1 arguments.

$$\widehat{\varphi}_{g(e,a,b)}(f,x) = \begin{cases} \widehat{\varphi}_e(\tau^{\sim 0}, x) & \text{if } (i)\ \forall \langle \sigma, y\rangle \le a\ [\widehat{\Phi}_e(\sigma^{\sim 0}, y) \le T(\sigma^{\sim 0}, y) + b] \\ & \quad (ii) \forall \langle \sigma, y\rangle \le \langle \tau, x\rangle\ \ [\forall \eta \subseteq \sigma(a < \langle \eta, y\rangle) \\ & \qquad \Rightarrow \widehat{\Phi}_e(\sigma^{\sim 0}, y) \le T(\sigma^{\sim 0}, y)], \\ & \text{where } (\tau, x) \text{ is a locking fragment of } T \text{ on } (f, x); \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Next, we shall argue: (i) If $F \in \mathbf{C}(T)$, there exist $e, a, b \in \mathbf{N}$ such that $F = \widehat{\varphi}_{g(e,a,b)}$. (ii) For every $e, a, b \in \mathbf{N}$, we have $\widehat{\varphi}_{g(e,a,b)} \in \mathbf{C}(T)$. Due to the space constrains, we omit the detailed argument. □

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

Here we demonstrate the type-2 analogs of three interesting complexity theorems in classical complexity theory: Rabin's theorem [17], recursive relatedness theorem [2], and Gap theorem [3]. Our purpose is to show that our type-2 asymptotic approach is a reasonable one that can lead to a full scale investigation of type-2 complexity theory. Due to the space constraints, detailed proofs are removed.

A technique using diagonalization together with a priority argument with no *injury* is also known as cancellation argument [16], by which Rabin proved that, for any recursive function t, there is a recursive 0-1 valued function f such that $f \notin \mathit{DTIME}(t)$. We modify Rabin's proof and obtain an analogous type-2 result as follows. The proof is also given in the full version of this paper.

THEOREM 17 (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 $F \notin \mathbf{C}(T)$.*

As we mentioned earlier, recursive relatedness is a bridge for complexity theorems between different complexity measures [2]. A type-2 analog will be also essential if we want to further abstract away from a particular model of type-2 computation. We thus formulate a type-2 Recursive Relatedness Theorem in the following. We omit the proof since it can be obtained from the original proof with some minor modification.

THEOREM 18 (TYPE-2 RECURSIVE RELATEDNESS THEOREM) *For any two complexity measures for OTM's, $\langle \Phi \rangle_{i \in \mathbf{N}}$ and $\langle \Psi \rangle_{i \in \mathbf{N}}$, there is a computable functional $R : \mathcal{T} \times \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ such that, for every $i \in \mathbf{N}$,*

$$\Phi_i \ \le_2^* \ \lambda F, x \cdot R(F, x, \Psi_i(F, x)) \ \textit{and} \ \Psi_i \ \le_2^* \ \lambda F, x \cdot R(F, x, \Phi_i(F, x)). \quad \square$$

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 [5, 24] 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 [11]. For simplicity, here we restrict the gap factor to recursive functions. We obtain the following result.

THEOREM 19 (TYPE-2 GAP THEOREM) *For any 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 [11]. In other words, the cost of a query is at least the length of the answer. This model is called *Answer-Length-Cost Model* [19]. We use the following computable predicate, $P(f, x, k)$, to determine the value of $T(f, x)$.

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

In 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 $\mathsf{dom}(\sigma) \subseteq \{0, 1, \ldots, x\}$, then so does $g(T(f, x))$ and no OTM e that queries beyond $\mathsf{dom}(\sigma)$ can have $\widehat{\Phi}_e \leq_2^* g \circ T$. □

6. Conclusion

As a matter of fact, a general type-2 complexity theory is still an unclear territory. Many applications of type-2 (or higher) computations (e.g., machine learning, interactive computing, real computation, and the theory of programming languages) use their own approaches to address their complexity issues. It is usually difficult to apply one approach that is developed for one particular application to another application. 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 $\leq_2^*$ can provide such a step towards a general theory of type-2 complexity.

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

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

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

[6] 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.

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

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

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

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

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

[12] 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.

[13] 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.

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

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

[16] 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.

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

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

[19] 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.

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

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

[22] 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.

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

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