Type-2 Complexity Theory

by

Chung-Chih Li

Abstract of Dissertation

April, 2001

The notion of type-2 computability occurs naturally in many practical and theoretical settings in computer science. For examples, machine learning, programing languages, databases enquiry, complexity-theoretic problem reductions, and so on, are immediate applications of type-2 computation. However, there is no satisfactory type-2 complexity theory to characterize the computational cost of these widely ranged applications. Thus, the purpose of this thesis is to give a theoretical framework for analyzing the complexity of type-2 computation.

We use the Oracle Turing Machine (OTM) as our standard formalism for type-2 computation. The best way to characterize the computational cost of type-2 computation is to give a robust notion of type-2 complexity classes. In order to do so, we first study the induced topologies determined by type-2 continuous functionals of type $(\mathbf{N} \to \mathbf{N}) \times \mathbf{N} \rightharpoonup \mathbf{N}$. Then, based on the compact sets in the induced topologies, we define a type-2 almost-everywhere relation $\leq_{\mathbf{2}}^*$ over type-2 continuous functionals. The type-2 almost-everywhere relation $\leq_{\mathbf{2}}^*$ provides an analogous notion of asymptotic approach for complexity analysis in type-2. We also specify a clocking scheme for

OTMs based on a class of computable functions called *Type-2 Time Bounds* ($\mathbf{T_2TB}$). With the tools we developed, each type-2 time bound $\beta \in \mathbf{T_2TB}$ determines a type-2 complexity class $\mathbf{C}(\beta)$. We also define a type-2 big-O notation – $\mathbf{O}(\beta)$ – which would be a useful tool for type-2 algorithm analysis.

To justify our notion of type-2 complexity classes, we prove the Union Theorem, the Gap Theorem, the Compression Theorem, and the Speed-up Theorem in type-2 along the lines of classical complexity theory. Most of the theorems we proved are very different from their type-1 counterparts. We thus learn that the structure of type-2 complexity classes is not as sturdy as the structure in type-1; they are very sensitive to the topological constraint. With theses complexity results, we have a reasonable outlook for a general type-2 complexity theory.

Keywords: Complexity Theory, Computational Complexity, Complexity Class, Topology, Type-2 Functional, Continuous Functional, Type-2 Time Bound, Oracle Turing Machine.

Type-2 Complexity Theory

by

Chung-Chih Li
B.S. Tamkang University, Taiwan, 1986
M.S. Lamar University, Texas, USA, 1991

Dissertation

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer and Information Science
in the Graduate School of Syracuse University

April, 2001

Approved ______________________________
Date ________________________________

Contents

List of Tables

List of Figures

Preface

As the computational complexity theory in type-1 has become a well-studied branch of theoretical computer science, there is still no satisfactory answer to Constable's old question raised in [Con73]: what should be the type-2 complexity theory? Thus, the purpose of this thesis is to try to give a clearer picture of a general type-2 complexity theory as a partial answer to Constable's question in [Con73]. Following Constable's paper [Con73], there have been two possible approaches in studying higher type complexity theory. One of the two is the programming-language approach initiated by Constable's analysis on his programming language **PL** of [CG72]. The other one is an axiomatic approach along the lines of Blum's plausible two axioms for type-1 complexity theory. Constable gave a third axiom in that 1973 paper. Then, Mehlhorn took up the problem and proposed his own programming language $\mathcal{L}()$ and six axioms in [Meh76] for type-2 complexity theory. The core question in Constable's mind is that: What should a type-2 complexity theory be? Almost three decades later, we still do not have a satisfactory notion of type-2 complexity classes to address the question.

We are motivated by Constable's unanswered question. However, neither of the two approaches mentioned above is adopted in our research. Because, as we believe, in the study of higher typed computational complexity, machine characterizations, if it is possible, can give a more intuitive and understandable notion to conceptualize complexity classes than other approaches. Therefore, we use the Oracle Turing Machine (OTM) model, a great antiquity introduced by Turing [Tur39][1] himself in the study of relative computability, as our type-2 computing apparatus and the number of steps it performs as our complexity measure in our research.

We present a detailed clocking scheme for OTMs based on Cook and Kapron [CK89, CK90, KC91] and Seth's [Set94] idea. We also define a special class of type-1 functions called *Type-2 Time Bounds* (denoted as $\mathbf{T_2TB}$) and prove that our type-2 time bounds are suitable to be used as clocks in our clocking scheme. Every $\beta \in \mathbf{T_2TB}$ determines a *limit functional* and has some special properties that are suitable for our purposes. Based on these setups, we are able to define type-2 complexity classes. We denote our type-2 complexity class by $\mathbf{C}(\beta)$ which is determined by a $\beta \in \mathbf{T_2TB}$. Such notation reflects the same idea involved in *DTIME*(t). The proofs of many nontrivial complexity theorems need to loose the definition of $\mathbf{C}(\beta)$ in a way that we allow a reasonable small amount of exceptional inputs on which the computational cost of an $F \in \mathbf{C}(\beta)$ needs not to be bounded by β. Such toleration inevitably leads us to the study of the topological nature of type-2 computation. We come up with an induced topology determined by a concerned type-2 computable functional and show that a compact set in such a topology can be accepted as a "small amount" of exceptions. Since such a compact set can only conduce to finitely many results of the computation of the concerned functional, it can be taken as an analogy to the notion of "finitely many type-0 inputs" in type-1 computation.

In [Set94] Seth suggested a notion of machine based type-2 complexity classes similar to ours. Also, he speculated that some classical complexity results such as the Gap, Union, and Speed-up Theorems can be obtained in type-2. However, after

[1]Turing called such machines o-machines. [Tur39] is collected in [Dav65]

we closely examine the idea and give a precise definition of type-2 complexity classes, we find that not every type-1 complexity theorem can be easily obtained by lifting its original proof. We thus understand that the computational behaviors in type-1 and type-2 are very different. Therefore, we prove our own versions of type-2 Union, Gap, Compression, and Speed-up Theorems. We also point out that a type-2 analog of the Honesty Theorem is of no importance in type-2 computation.

The original Constable's question posed in [Con73] is to ask what a class of type-2 feasible functionals should be. **BFF** seems to be the most widely accepted notion for type-2 feasible functionals. We point out that **BFF** fails to be a type-2 complexity class under our notion. However, since type-2 computation by no means is parallel to type-1, we believe that we should be more liberal here and consider **BFF** as a right notion of complexity class in which each functional shares the same complexity property. In fact, **BFF** can be represented as a union class, $\mathbf{C}(\langle \beta_i \rangle)$. That is, although **BFF** cannot be determined by a single type-2 time bound, **BFF** can be determined by a computable sequence of type-2 time bounds.

The number of oracle queries made during the course of the computation is a special complexity measure in type-2 computation. In many applications such as machine learning, the query-complexity has been seriously considered. However, query-complexity fails to meet Blum's two axioms, which adds to the difficulty of the study. Nevertheless, we gave three notions of query-optimum programs to see how well we can have an optimum program in principle. A few phenomena in query-complexity have studied, which can be seen as type-2 speed-up phenomena associated with query-complexity.

Acknowledgements

This is certainly a great moment for me to press my deep thankfulness to many people involved, either directly or indirectly, in this work. First of all, I would like to thank my advisor Prof. James S. Royer for his super advising and unfailing help throughout my study. It was Prof. Royer who introduced me to this wonderful field of theoretical computer science, especially recursion theory and complexity theory. He initiated the original idea of this thesis and encouraged me with confidence to push the limit.

I would like to thank Prof. Ernest Sibert, Kishan G. Mehrotra, Howard A. Blair, and Susan B. Order, for showing interest in my work and giving me valuable comments. I would also like to thank Prof. Chung Chen for chairing my defense despite his already busy schedule.

I am grateful to the Department of Elec. Eng. and Computer Science, Syracuse University, for the support throughout these years. Working with Prof. Hartmann R.P. Carlos, Prof. Mehrotra, and Prof. Blair in their mathematical courses as a teaching assistant has been a pleasant experience in my graduate career. The duty gave me a great opportunity to strengthen my skills from their professionalism in many ways, mathematically and philosophically. This work was also supported in part by NSF grant CCR-9522987.

Last but not least, I would like to thank my family with all my heart. My parents gave me unlimited support as if I was a college boy living in a dormitory, even though I'm a father of two and living overseas far away from home. Thank you, Dad and Mom!! To my wife, Shih-Wei, who supported me in whatever I wanted to do, and my kids, Sean and Leon, who encouraged me by showing their pride in me and hugging me when I got home, I thank and love you all.

Chapter 1

Introduction

In the past few decades, computational complexity theory has become a mature and well-studied branch of mathematics. For most of its development the theory has focused on type-1 computation. In part this was because the obvious applications of computational complexity were type-1 problems. However, another part of the reason was because there were no well-understood machine/cost models for higher type computation.

In [Tur39] Turing himself gave an excellent type-2 machine model–the Oracle Turing Machine (OTM)–and there has been quit a bit of work on higher type *computability*.[1] However, higher-type complexity issues were largely ignored. Moreover, the scattered literature that deals with higher-type complexity has focused on *feasibility*. For example, Constable [Con73] raised the issue of developing a general higher-type complexity theory, but the work ended up with defining a type-2 analog of polynomial-time computable functions. Similarly, Cook and Kapron's research [CU89, CK89, CK90, KC91, Coo91, KC96] on higher types concentrated on characterizing *feasible functionals.*

[1]See [Rog92] and [Odi89] for surveys of this work.

Key Problems For A General Type-2 Complexity Theory

Among the difficulties in extending complexity theory to handle type-2 computation is that there are no widely accepted answers to the following basic questions.

1. What are appropriate resource-bound functions for type-2 computation?

2. What is the size of a type-2 problem? For example, by reasonable conventions, we know the meaning of the size of an array or the size of an instance of the Traveling Salesman Problem. Clearly, to parameterize the problem size is indispensable in computational complexity analysis. But what is the size of a function? How should one measure an infinite object?

3. What is the proper type-1 analog of the concept of "finitely many type-0 objects"? In type-1 computation, we usually consider "finitely many" type-0 objects (e.g., natural numbers) as "few" or "ignore-able," because any program with something wrong on some finitely many type-0 inputs can be manually patched. But a finite set of functions cannot be examined by any program, since it is not "finite" at all.

The questions above are partially addressed in some works (e.g., [CK89] and [Set94]) but their answers, arguably, do not suffice for a more general type-2 complexity theory. Therefore, a general type-2 complexity theory needs to begin by trying to provide satisfactory answers to the questions above.

In the following we briefly discuss some important applications of type-2 computation, and then we formalize our Oracle Turing Machines as our standard machine model for type-2 computation. The type-2 complexity theory to be developed in this thesis will be based on this machine model. We also highlight some important works in this area and point out those that influenced our work. The final section of this chapter summarizes the work in this thesis.

1.1 Type-2 Computation

Typically, we consider natural numbers to be the only type-0 objects. Let $\mathbf{N}$ denote the set of natural numbers. Type-1 objects are maps of the form $\mathbf{N}^k \rightharpoonup \mathbf{N}$, for $k \in \mathbf{N}$. Type-2 objects are maps of the form:

$$(\mathbf{N}^{k_1} \rightharpoonup \mathbf{N}) \times (\mathbf{N}^{k_2} \rightharpoonup \mathbf{N}) \times \cdots \times (\mathbf{N}^{k_m} \rightharpoonup \mathbf{N}) \times \mathbf{N}^n \ \rightharpoonup \ \mathbf{N},$$

where $k_1, k_2, \dots, k_m, n \in \mathbf{N}$. However, for simplicity, in this thesis we shall restrict our attention to type-2 objects of the form $(\mathbf{N} \to \mathbf{N}) \times \mathbf{N} \rightharpoonup \mathbf{N}$.

A fundamental difference between ordinary and higher type computability is that there is no higher-type analog of Church's Thesis. There are quite reasonable, but incompatible, paths one may follow in formalizing a notion of higher-type computability. However, at type-level 2, these differences are not so far apart, and there turn out to be many interesting (and unproblematic) applications of type-2 computability. This is a large part of why we choose to focus on just type-2 computation in this thesis.

1.1.1 Examples of Type-2 Computation

Here we briefly describe a few typical examples of type-2 computation.

Programing Languages: In computer science, the most natural and common examples of higher-type computation are programming languages that allow the use of higher-type parameters; even early versions of FORTRAN permitted such things. For example, consider the following Haskell program:

```
sum :: (Int → Int) → Int → Int → Int
sum f a b =
   if a > b
      then 0
      else (f a) + sum f (a + 1) b
```

This program determines the functional $\Psi(f, a, b) = \sum_{x=a}^{b} f(x)$. In fact, programming languages have been an important tools in the study of higher-type computational complexity.

Cook Reductions: Cook reductions were introduced by Cook himself in order to study the reducibility of NP problems [Coo71, GJ79]. For example, We denote

$$\text{Traveling Salesman Problem} \quad \propto \quad \text{SAT} \tag{1}$$

to mean that the *traveling salesman problem* (TSP) can be *reduced* to the *satisfiability* (SAT) problem via some polynomial-time computable Turing Machine with a SAT oracle. That is, there is an OTM, M^{SAT}, such that, for all $x \in \mathbf{N}$,

$$M^{\text{SAT}}(x) = \begin{cases} 1 & \text{if } x \in \text{TSP}; \\ 0 & \text{otherwise,} \end{cases}$$

where the run time of computing $M^{\text{SAT}}(x)$ is bounded by a fixed polynomial in terms of $|x|$, where $|x|$ is the size of the instance x. Formally, Cook reductions are known as *polytime Turing-reductions*. To see that this is indeed a type-2 computation, we consider the oracle attached to the OTM, M, as its type-1 input. Thus, the OTM M computes a functional $F : 2^{\mathbf{N}} \times \mathbf{N} \rightharpoonup \{0, 1\}$ such that, for every $x \in \mathbf{N}$,

$$F(\text{SAT}, x) = \begin{cases} 1 & \text{if } x \in \text{TSP}; \\ 0 & \text{otherwise.} \end{cases}$$

In other words, the function, $\lambda x.F(\text{SAT}, x)$ is a characteristic function of TSP. In the model above, we first fix the type-1 input to a specific set (in our example, SAT) and then we consider the computational complexity issue in terms of the type-0 inputs. Karp [Kar72] reduced Cook reductions to many-one reductions via polynomial time computable functions. Although Karp reductions are pedagogically more intuitive reductions in the context of NP problems, the connection to type-2 computation becomes not explicit.[2]

Enumeration Operators: Friedberg and Rogers [FR59] introduced *enumeration reduction* in order to study the reducibility between sets of natural numbers. Following Rogers' notations in [Rog92], $A \leq_e B$ denotes that A can be enumeration reduced to B. For the Turing reduction, we use $A \leq_T B$. *Enumeration operators* are simply mappings from $2^{\mathbf{N}}$ into $2^{\mathbf{N}}$, but they provide a strong notion of reductions in comparison with Turing reductions. That is, $A \leq_e B$ implies $A \leq_T B$ but the inverse fails. This is because, in the context of Turing reductions, a B-oracle can decide $x \in B$ both positively and negatively. But, in the context of enumerate reductions, only a positive answer to the query $x \in B$ can be obtained.

Intuitively, $A \leq_e B$ means that, if we are given an enumeration of B, then we can effectively enumerate A. In other words, if $A \leq_e B$, then we should be able to list any fixed $x \in A$ based on a finite portion of the information of B. Let us consider an enumeration operator Γ defined by:

$$\Gamma(B) = \left\{ a + b \;\middle|\; a \neq b \;\wedge\; a, b \in B \right\}.$$

Let $B = \{1, 2, 3, 4\}$ and $\Gamma(B) = A$. Thus, $A = \{3, 4, 5, 6, 7\}$. One can see that we can have a recursive φ-program e such that, for every $S \subseteq \mathbf{N}$, if we give an enumeration

[2]For more details about NP problems and Cook reductions, see [HU79, GJ79]. For Turing-reductions and many-one reductions, see Chapter 9 of [Rog92].

of set S as inputs of e, then we can have an enumeration of $\Psi(S)$ as the outputs of e. For example,

$$\begin{aligned}
\varphi_e(\langle\emptyset\rangle) &= \langle\emptyset\rangle, \\
\varphi_e(\langle\{1\}\rangle) &= \langle\emptyset\rangle, \\
\varphi_e(\langle\{1,2\}\rangle) &= \langle\{3\}\rangle, \\
\varphi_e(\langle\{1,2,3\}\rangle) &= \langle\{3,4,5\}\rangle, \\
\varphi_e(\langle\{1,2,3,4\}\rangle) &= \langle\{3,4,5,6,7\}\rangle,
\end{aligned}$$

where $\langle\{\cdots\}\rangle$ is a canonical indexing for finite subsets of $\mathbf{N}$. We say such an effective procedure e determining the enumeration operator Ψ_e. In the example above, we have $\Psi_e = \Gamma$ and $\{3,4,5,6,7\} \leq_e \{1,2,3,4\}$.

Note that every partial function can be represented as a *single-valued* set in $\mathbf{N}\times\mathbf{N}$. Thus, if Ψ is an enumeration operator such that, for every single-valued set S, $\Psi(S)$ is also as single value set, then Ψ determines a functional of type $(\mathbf{N} \rightharpoonup \mathbf{N}) \to (\mathbf{N} \rightharpoonup \mathbf{N})$.

Moreover, if Φ_z is an enumeration operator such that, for every single-valued set S, $\Phi_z(S)$ is also single valued, then Φ_z determines a recursive operator $\Psi : (\mathbf{N} \rightharpoonup \mathbf{N}) \to (\mathbf{N} \rightharpoonup \mathbf{N})$. Such enumeration operators are called *recursive operators*.[3]

Machine Learning: Arguably, learning can be considered as a process of observing an unknown phenomenon or concept through examples and conjecturing a hypothesis for the phenomenon or concept. Since a concept in the domain of interest usually consists of infinitely many examples, such a learning process is best described as a type-2 problem. For *machine learning*, we focus on the study of how a machine can successfully learn a target concept and, of course, of its limitation and cost. A learning

[3]For more details about enumeration reductions and recursive operators, see Chapters 9 and 15 of [Rog92] or Chapters II of [Odi89] and XIV of [Odi99].

machine must be constructed in terms of some computing formalism such as Turing Machines or algorithms in an *acceptable* programming language. Thus, the theory of machine learning is also know as *Formal Learning Theory*.

In [Gol67] Gold established a theoretical framework for mechanically identifying a language based on some observed sentences of the language. A language is represented by a subset of $\mathbf{N}$, and sentences of the language are simply the members of the set. An external agent who knows the language provides sample sentences to the learning machine. Many machine learning models follow Gold's paradigm. For example, the Inductive Inference Machine (IIM) is a direct implementation of Gold's paradigm for identifying a concept *in the limit*. Since IIMs (or simply *inductive inference*) provide a general framework for studying machine learning, they have been extensively studied over recent decades (e.g., [BB75, AS83, CS83, Cas86, JORS99] or Chapter VII.5 of [Odi99]). Typically, an IIM M is a Turing Machine that works as follows:

1. We fix an external agent that carries a certain concept (e.g., a total function).

2. M starts from an initial state with the input and output tapes empty.

3. From time to time, one of the following situations happens:

   (a) M enters a special state in which M holds its execution and waits for some inputs from the external agent. After the input has been prepared on the input tape, M resumes and proceeds to the next state.

   (b) M starts to write a number on the output tape. When it finishes the writing, M either simply stops or continues its execution.

   The two situations described above may repeatedly happen, that is, M may keep asking for inputs and changing the contents of the output tape.

For example, the external agent may carry a concept of total function $f : \mathbf{N} \rightarrow \mathbf{N}$, and it provides instances to M in a sequence, $f(0), f(1), f(2), \ldots,$ one at a time. Eventually, M outputs a number i and stops or M will not output another number other than i. In either case, we say that M converges to i on f (denoted by $M(f) \downarrow= i$), otherwise we say that M diverges on (f) (denoted by $M(f) \uparrow$). Clearly, M determines a functional $F_M : (\mathbf{N} \rightarrow \mathbf{N}) \rightharpoonup \mathbf{N}$ such that,

$$F_M(f) = \begin{cases} i & \text{if } M(f) \downarrow= i; \\ \text{undefined} & \text{if } M(f) \uparrow . \end{cases}$$

In the example above, if i is a program for f (i.e., $\varphi_i = f$ where φ is an acceptable programming system), then we say that the IIM, M, identifies f. If S is a set of functions and, for every $f \in S$, M can identify f, then we say that M identifies S. Moreover, if there is an IIM to identify S, we say that S is *identifiable*. For example, the set of *primitive recursive* functions is identifiable but the set of *recursive* functions is not [BB75]. Changing the criteria of identification may result in different identifiable classes [CS83]. For example, we can replace the requirement, $\varphi_i(x) = f(x)$, by $\varphi_i(x) =^* f(x)$ where some finite disagreement between φ_i and f is allowed.

In fact, receiving example from a target concept in a passive manner does not reflect a more general notion of learning. Thus, a prodigious amount of variants and generalizations of this simplest learning model have been developed and well-studied in the active field of *Computational Learning Theory*. For example, the *Student-Teacher* model[4] allows the student to express different kinds of questions to the teacher who knows the target concept f. Since the student's questions may vary from a simple one like "*does* $f(x) = y$*?*" to a highly undecidable one like "*is there an* x *such that* $f(x) = 0$*?*" or "*does* $\varphi_t = f$*?*". In this model, an f-oracle no longer suffices to be a

[4]More details about the Student-Teacher learning model can be found in [Ang88, GS92] and the book [JORS99].

teacher. Thus, a higher-type oracle is needed for the more general Student-Teacher learning model.

1.1.2 Complexity Issue of Type-2 Computation

Clearly, the complexity issue should be addressed in studies of the above applications. However, there are not many results in this area. For example, higher-type computation in programming languages has existed for a long time, but almost the entire body of algorithm analysis is restricted to the study of type-1 complexity. This is simply because there is no satisfactory theoretical framework for analyzing higher-type computation.

For Cook reductions, it is explicitly required that the reduction has to be polynomially computable in terms of the instance size. In such a way, the cost of the reduction does not become a significant part of the cost of computing the reduced problem when we replace the oracle by a Turing Machine that computes the oracle. (In our example on page 4, the SAT oracle can be replace by a Turing Machine that decides SAT.) However, if we consider Cook reductions from the type-2 point of view, the oracle of M needs not to be fixed to SAT. Thus, the complexity issue is only partially addressed, because we do not know (or say, we do not care in the context of Cook reductions) how M^O computes if O is an oracle other than SAT. To fully understand the behavior of an OTM indeed demands a more general complexity theory in type-2.

In *Computational Learning Theory*, complexity turns out to be an important issue. We certainly want to know which concepts are easy to learn and which are not. For example, Jain and Sharma in [JS94] used the notion of reduction to compare the difficulty of identification between different classes of languages. Some earlier works such as [DS86] also addressed the complexity issue in great detail. However, complexity measures for machine learning are rather diverse. A few common measures are:[5] (*i*) the number of times the learner changes its conjecture, (*ii*) the number of times the learner asks for inputs, and (*iii*) the time the learner needs for computing before it converges to a correct conjecture. These ad hoc attempts seem to be necessary for modeling the computational complexity of learning machines not only because we have different considerations in different learning models, but also because a general type-2 complexity theory does not yet exist.

Therefore, part of the motivation for this thesis is to work toward placing the above ad hoc complexity results in some unified framework.

1.2 Background Literature on Type-2 Complexity Theory

Hartmanis and Sterns's paper [HS65] marked the start of the systematic study of the complexity of computable functions. They used multiple-tape Turing Machines as the standard computing model and time as the complexity measure to explore a few computational complexity properties. Hartmanis and Sterns defined a set of T-computable functions as the collection of computable functions with computational complexity bounded by the monotone-increasing computable function $T : \mathbf{N} \to \mathbf{N}$. This definition was the first precise definition for complexity classes in terms of Turing Machines. Many complexity results in this paper are now taken for granted in computer science such as the Linear Speed-up Theorem. Two years later, Blum [Blu67] moved one step forward. Blum abstracted the computational complexity of a computable function away from any specific computing model. He introduced two

[5] See [JORS99] for details.

axioms[6] that intuitively appeared broad enough to encompass many reasonable complexity measures in real computing. Following Blum's lead, numerous researchers established many machine independent complexity results.[7] This work provided a rich chapter of computational theory.

From these beginnings, computational complexity theory has grown into a deep and broad-ranging research field. We refer the reader to Papadimitriou's text [Pap94] for a comprehensive survey.[8] In all this work, however, work on type-2 complexity has remained a tiny thread. We next survey this work, starting with Constable's 1973 paper [Con73] which seems to be one of the first places the issue was raised.

1.2.1 Syntactic Approaches

In [Con73] Constable used a programming-language approach to study the computational complexity of type-2 functionals. Based on the multi-scheme language of his previous work [CG72], Constable defined a programming language called **PL** for type-2 procedures. **PL** provides *parallel assignments* by using *semaphore variables* for synchronization in a conventional way of parallel programming. While parallel computations were allowed in **PL**, Constable conceived that the result of any type-2 computing should not depend on the cost of computing its type-1 inputs. Thus, for a type-2 parallel procedure to be considered as a type-2 functional, it has to be *extensional* on its type-1 inputs.[9] Then, he pointed out that the collection of all *extensional* parallel procedures in **PL** coincided the class of Rogers' *enumeration operators*

[6] We present the two axioms in Definition 90 on page 118 where the Blum complexity measure is defined.

[7] In Chapter 5 we will specify some theoretically important results that we are interested in.

[8] Wagner's [WW85], Calude's [Cal88], and Odifreddi's [Odi99] books are all excellent textbooks of computational complexity theory. The three textbooks treat the subject more from the recursion-theoretic aspect.

[9] In Constable terminology, a type-2 parallel procedure whose results do not depend on the cost of computing its type-1 inputs is called *determinate*.

in [Rog92].

However, Constable also determine that as far as the computational complexity was concerned, the cost of computing type-1 inputs should be taken into account.[10] Thus, in addition to Blum's two axioms, Constable gave a 3rd axiom that enforced the relationship between the complexity of type-2 functionals and the cost functions of their type-1 inputs. Time and space meet the three axioms and thus are two appropriate *Complexity Measures* for type-2 computations. The three axioms appeared to be able to serve as the basis for an abstract type-2 computational complexity theory along the line of Blum's axiomatic approach to type-1 computational complexity theory. Constable also lifted the proof of the Operator Speed-up Theorem [MF72] in accordance with his setup and showed that the speed-up phenomenon still exists in type-2. The paper ended up with a definition of Polynomial Operators. The class of Constable's Polynomial Operators was studied and given another characterization by Mehlhorn in later study.

In [Meh74, Meh76] Mehlhorn generalized Cook reductions between sets to apply to functions of type $\mathbf{N} \to \mathbf{N}$. In order to understand these generalized feasible computable reductions in terms of type-2 functionals, Mehlhorn followed Constable's approach of [Con73] and syntactically defined a class of type-2 functionals named $\mathcal{L}()$. Intentionally, $\mathcal{L}()$ was defined to be a type-2 analog of Cobham's $\mathcal{L}$ [Cob65] by introducing type-1 variables to $\mathcal{L}$. Since Cobham's $\mathcal{L}$ was the same class of type-1 polynomial time computable functions (denoted by *PTIME*), $\mathcal{L}()$ was expected to be the type-2 analog of *PTIME*. A few properties of $\mathcal{L}()$ such as *honesty* were proved in

[10] Note that [Con73] restricted type-2 functionals to *partial recursive operators* defined by **PL**. Thus, the type-1 inputs of a **PL**-computable functional must be some other **PL** codes, and hence they are also **PL**-computable. Moreover, each type-1 input was explicitly associated with a time function that played a crucial role in the 3rd axiom of Constable's axiomatic setup for type-2 complexity. In such a way, the cost of computing the type-1 input whenever needed is counted as a part of the overall computational cost. This approach is different from our OTM model where the oracle may represent an uncomputable function and will give any needed value for free.

[Meh74, Meh76] as evidence that $\mathcal{L}()$ is a sensible notion of feasible type-2 functionals. In fact, $\mathcal{L}()$ turned out to be the same class as Constable's Polynomial Operators, which is now known as **BFF**.[11]

Few works in this area were published in the decade following Mehlhorn's papers. Moreover, the scattered works done during this period were all based on syntactic (programming) approaches which, arguably, seemed to be more intuitive for describing higher-type computations. There was really no work directly related to our present approach until late 80's when Cook and his colleagues joined the study.

In [CU89] Cook and Urquhart defined the system PV^ω to simplify Buss's IS_2^1 [Bus86] for the class of polynomial time functionals at all finite types. The system PV^ω was defined based on the *simply typed* λ*-calculus* and the system PV introduced in [Coo75]. Then, in [CK89, CK90, KC91] Cook and Kapron extended the work of [CU89] and examined Mehlhorn's $\mathcal{L}()$. As in Mehlhorn's work of [Meh74, Meh76], Cook and Kapron extended Cobham's syntactic definitions for $PTIME$ to finite-type functionals and obtained a simple typed programming language called *Bounded Typed Loop Programs* (**BTLP**). **BTLP** described a class of finite-type functionals, and this class, for the first time, was called the *Basic Feasible Functionals* (**BFF**'s). The type-2 section of **BTLP**[12] turned out to be the same class of Mehlhorn's $\mathcal{L}()$. We denote this type-2 class by $\mathbf{BFF}_2$.

In [Coo91] Cook presented a functional L which is not in **BFF** but is, in Cook's opinion, intuitively feasible. Thus, Cook contended that **BFF** does not include all intuitively feasible functionals and proposed three conditions for a class of functionals being feasible. That is, the class of functionals P is feasible if:

[11] **BFF** stands for (the class of) Basic Feasible Functional(s) which is now considered as a feasible class for higher-type functionals. We will come to **BFF** in a moment.

[12] The type-2 section of **BTLP** is the class of type-2 functionals that can be computed by some **BTLP**-programs.

1. **BFF** $\subseteq P$.

2. P must be closed under λ-abstraction and application.

3. Every type-2 functional $F \in P$ must be POTM[13] computable.

Cook proved that L satisfies condition 3, and thus the λ-abstraction and application closure of $\mathbf{BFF}\cup\{L\}$ satisfies the above three conditions. However, Seth [Set93] did not think Cook's functional L should be considered as a feasible functional and proposed another condition. We will describe Seth's point in a bit more detail in the next subsection, where we briefly discuss the machine characterization approach.

A recent work of Irwin, Kapron, and Royer [IKR01a] provided another programming formalism, *type-2 Inflationary Tiered Loop Programs* ($\mathbf{ITLP}_2$ for short), to characterize $\mathbf{BFF}_2$ without machines and clocks involved. There is a simple and low complexity translation between the two programming formalisms: $\mathbf{ITLP}_2$ programs and polynomially-clocked OTMs.[14]

Most researchers working in higher-type complexity theory have chosen syntactic approaches for their studies; for example, see Townsend [Tow90], Bellantoni [BC92, Bel92], and Leivant [Lei95]. This is partly because programming languages that allow passing program code as parameters catch an intuitive idea of higher-type computation. Another reason is that it is rather difficult to formalize machines for type-3 or higher functionals.

[13] POTM stands for Polynomial-time computable OTM. Since POTM plays an important role in the development of our work, we will discuss it with more details in Section 1.3.

[14] We will discribe how to clock an OTM in Section 1.3. And, after we formally define our type-2 time bounds in Chapter 3 we will formalize our clocking scheme in Section 3.4.

1.2.2 Machine Characterizations

In type-1, machine characterization seems to be a natural approach to study the complexity of computable functions. In doing so, we do not lose generality: the well-known Church's Thesis states that Turing machines describe the same class as other formal systems such as Herbrand-Gödel's recursive functions and Church's λ-terms do. But this is not the case for the study of higher-type computable functionals. Any machine that goes beyond OTM seems to have some difficulties and limitations. For example, Kleene in [Kle60, Kle62] extended the original OTM to a machine model equipped with higher-type oracles for computable functionals of all finite types. But Kleene's model inevitably needs to restrict oracle queries to questions that are computable from the inputs. In other words, if we need to query the oracle at argument α, we have to prepare an index of α to the oracle. Kapron [Kap91] noted the difficulties of this model when feasibility was concerned, because a query with arbitrary indices can result in a very complicate computation. These difficulties were also discussed in detail in [Set94] and in a recent work [IKR01b].

In [CK89] Cook and Kapron defined **BFF** and gave a programming formalism **BTLP** to characterize **BFF**. A few approaches for studying higher-type computability and complexity are reviewed in [Coo91]. In the same paper, Cook pointed out that **BFF** might have missed some intuitively feasible functionals. He thus proposed three conditions (see page 13) for any feasible class to hold. Afterwards, Kapron and Cook decided to consider an even more intuitive notion to understand type-2 feasibility. Thus, in [KC91, KC96] Kapron and Cook algebraically defined second-order polynomials that took type-1 functions and natural numbers as their input arguments (see Definition 1 on page 19). They also defined the length of total functions (see Definition 2 on page 20). Each total computable function is associated with a length function that is also total computable. Then, *basic polynomial-time functionals* were defined as a class of type-2 functionals such that, for every F in that class, there exist a second-order polynomial $\mathbf{q}$ and an OTM that computes F within time bounded by $\mathbf{q}$ in terms of the length of F's type-1 inputs and the size of the numerical inputs. Then, they proved that $\mathbf{BFF}_2$ and basic polynomial-time functionals are exactly the same class. In such a way, Kapron and Cook had shown the great similarity between *PTIME* and $\mathbf{BFF}_2$.

Seth took up the feasibility problems for higher-type functionals. In his work of [Set94, Set95], Seth modified Kleene's original machine model for higher-type computable functionals. He defined *Higher Type Oracle Turing Machines* (HOTMs) for functionals of all finite types in order to give **BFF** a machine characterization. Although Seth's main concern was still with feasibility, we noted an emerging picture of a general computational complexity theory (at least at type-2) in his works. We briefly discuss this work the following paragraphs. More details related to our clocking scheme will be discussed in Section 1.3.

Seth examined **BFF** and reconsidered Cook's three conditions (see page 13). In [Set92] he presented a class of type-2 functionals to show that there is no largest recursively enumerable class of type-2 functionals which satisfies Cook's three conditions. Although Cook's functional L of [Coo91] did *preserve*[15] *PTIME*, which is the reason Cook considered L as an intuitive feasible functional, Seth pointed out that L failed to preserve the class of *elementary functions*,[16] which Seth viewed as a reasonable type-1 complexity class for feasible functionals to preserve. Thus, in

[15]Let $\mathcal{F}$ be a class of type-2 functionals of type $(\mathbf{N} \to \mathbf{N}) \times \mathbf{N} \to \mathbf{N}$ and $\mathcal{C}$ be a class of functions from $\mathbf{N}$ to $\mathbf{N}$. We say that $\mathcal{F}$ preserves $\mathcal{C}$ iff

$$\forall F \in \mathcal{F} \forall f \in \mathcal{C}\ [\lambda x F(f, x) \in \mathcal{C}].$$

[16]See Chapter VIII.7 of [Odi99] for details of elementary functions.

[Set93] Seth proposed another condition for feasible classes, namely that every functional in a feasible class should *preserve* any "reasonable" type-1 complexity class. Later on, Pezzoli [Pez98] modified Seth's preservation condition by restricting Seth's "reasonable type-1 complexity classes" to her "length function classes" and proved that $\mathbf{BFF}_2$ is the only class of type-2 functionals that satisfies Cook's three conditions plus her new condition.

All Seth's works were based on machine characterizations. Based on OTMs, he [Set94] proposed two approaches to define type-2 complexity classes analogous to the classical notion of type-1 complexity classes. That is, each complexity class was determined by a time bound.[17] Seth described a way to calculate time bounds for OTMs. We modified and implemented this idea in our clocking scheme. In fact, a clocking scheme was also implicitly involved in Kapron and Cook's work, but Royer was the first to explicitly describe a clocking scheme for OTMs [Roy97]. Royer defined polynomial-time effective operations and proved that they were more powerful on recursive functions than $\mathbf{BFF}_2$ under the assumption P = NP. Under a clocking scheme for OTMs, Royer also defined a class of functionals call *clocked basic polynomial-time functionals*. This class turned out to be the same class as Cook's *basic polynomial-time functionals* (i.e., $\mathbf{BFF}_2$). A polynomial-time analog of Kreisel-Lacombe-Shoenfield Theorem ([Rog92] page 362) was shown, which indicated that the different characterizations in [Roy97] all determined the same class, $\mathbf{BFF}_2$.

Kapron and Cook's definitions of second-order polynomials and length functions successfully characterized $\mathbf{BFF}_2$ and showed the great similarity between *PTIME* and $\mathbf{BFF}_2$. However, the cost of computing a length function, as we will see in next section, is not feasible at all. Thus, it is not suitable to directly use the length function in a clocking scheme for real applications. Thus, we develop our own scheme

[17]See Definitions 4 and 5 on page 21 for the two approaches.

for calculating time bounds in our work.

1.2.3 Axiomatic Approaches

The success of Blum's axiomatic approaches[18] influenced the study of the complexity theory in the 70's. It was quite natural for every complexity theorist who tried to lift the study to type-2 to speculate some sort of analogous axioms for type-2 complexity measures. For example, what really in Constable's mind was to extend Blum's two axioms for type-2 complexity theory. In [Con73] he proposed a third axiom based on the restriction that the type-1 input itself has to be computable. Then he went on to prove an Operator Speed-up Theorem based on his extended Blum axioms.

In fact, shortly before Constable's [Con73], Symes presented a work on type-2 complexity theory with an axiomatic approach [Sym71]. Comparing to Constable's proposal, Symes modified Blum's axioms in a more direct way. Symes's axiomatic system consists of Blum's first axiom and a modified Blum's second axiom. Since the OTM model used by Symes allowed a partial-function oracle, the computation of the OTM would be hanging up in case that the OTM said a query on which the partial-function oracle was undefined. Thus, a record of queries made during the course of the computation must be provided for the complexity measure in order to force the type-2 analog of Blum's second axiom to hold. With the two axioms, an abstract type-2 complexity measure based on OTMs is defined. Consequently, Symes proved Rabin's theorem, the Honesty Theorem, and the Speed-up Theorem.[19]

[18]See Chapter 5 for more details on Blum's axioms for complexity measures.

[19]For Rabin's theorem, see Section 4.2.4. For the Honesty Theorem and the Speed-up Theorem, see Chapter 5.

1.3 Kapron-Cook-Seth's Clocking Scheme

In type-1, we follow Hartmanis and Sterns's conventions in [HS65] to say that the complexity of computable function f is bounded by a total function t if and only if there is a *Deterministic* Turing Machine M such that M computes f and, on any input $x \in \mathbf{N}$, M halts within $t(|x|)$ steps, where $|x|$ is the length of the representation of x. We use $DTIME(t)$ to denote the class of computable functions that can be computed in this manner. If t is computable, then we can enumerate all computable functions in $DTIME(t)$ by clocking all Turing Machines with the clock, t.[20] That is, we simulate every Turing Machine M_e with index $e \in \mathbf{N}$ on input $x \in \mathbf{N}$ up to $t(|x|)$ steps. If the simulation does not halt within $t(|x|)$ steps, we simply terminate the simulation and output 0. On the other hand, if the simulation halts before the clock beeps, then we output the result of the simulation. We denote such t-clocked Turing Machine by $M_{e,t}$. Thus, if $f \in DTIME(t)$, then there is a Turing Machine M_e such that, $M_{e,t}$ computes f. If we let t range over all polynomials and take the union of $DTIME(t)$ for all such t, we obtain *PTIME*.

For type-2 computation, a direct analog is to lift the clock to something like a computable type-2 functional $\boldsymbol{t}$, and define a proper notion for the "length" of the type-1 object $|f|$. Then, we are able to say that "the computational complexity of the functional F is bounded by a total type-2 functional $\boldsymbol{t}$ if and only if there is an OTM M_e that computes F and on any input $f : \mathbf{N} \to \mathbf{N}$, M_e halts within $\boldsymbol{t}(|f|)$ steps". Kapron and Cook's [KC91, KC96] second-order polynomials and length functions are defined for this analog in the context of $\mathbf{BFF}_2$. The class of second-order polynomials is inductively defined by the following definition.

[20]Note that, this dose not immediately imply that $DTIME(t)$ is recursively representable. See Section 4.2.2 for more details on this issue.

Definition 1 (Second-order Polynomials [KC91, KC96]) *$\boldsymbol{q}$ is a second-order polynomial if and only if $\boldsymbol{q}$ is one of the following forms:*

$$c, \quad x, \quad \boldsymbol{q}_1 + \boldsymbol{q}_2, \quad \boldsymbol{q}_1 \cdot \boldsymbol{q}_2, \quad f(\boldsymbol{q}_1),$$

where $c \in \mathbf{N}$, x is a type-0 variable over $\mathbf{N}$, f is a type-1 variable over $\mathbf{N} \to \mathbf{N}$, and $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ are two second-order polynomials. ■

For example, $\boldsymbol{q} = f(f(x \cdot x)) + f(f(x) \cdot f(x)) + f(x) + 4$ defines a second-order polynomial with a type-1 variable f, a type-0 variable x, and a constant 4. Let $f(x) = x^2$. Then, $\boldsymbol{q}(f, 2)$ is calculated as follows.

$$\begin{aligned}
\boldsymbol{q}(f,2) &= f(f(2 \cdot 2)) + f(f(2) \cdot f(2)) + f(2) + 4 \\
&= f(f(4)) + f(2^2 \cdot 2^2) + 2^2 + 4 \\
&= f(4^2) + f(16) + 4 + 4 \\
&= 16^2 + 16^2 + 8 \\
&= 520.
\end{aligned}$$

It is clear that any second-order polynomial can be computed by some OTM, where the oracle is to provide needed values of the type-1 arguments. Thus, a second-order polynomial is a type-2 computable functional. Typically, the length of $x \in \mathbf{N}$, denoted as $|x|$, is simply the number of bits in the representation of x. For the length of a type-1 function, we have the following definition.

Definition 2 (Length Functions [KC91, KC96]) *Let $f : \mathbf{N} \to \mathbf{N}$. The length of f, also denoted as $|f|$, is the function $\mathbf{N} \to \mathbf{N}$ defined by*

$$|f| = \lambda n.\mathsf{max}\left(\left\{ |f(x)| \ : \ |x| \leq n \right\}\right).$$

■

Based on second-order polynomials and length functions, Kapron and Cook define a type-2 analog of *PTIME*.

Definition 3 (Basic Polynomial-time Functionals [KC91, KC96]) *Let F be a Type-2 functional. F is a basic polynomial-time functional if and only if there exist an OTM M and a second-order polynomial $\boldsymbol{q}$ such that M computes F and, on inputs $x \in \mathbf{N}$ and $f : \mathbf{N} \to \mathbf{N}$, M halts within $\boldsymbol{q}(|f|, |x|)$ steps.* ■

Then, Kapron and Cook proved that the class of basic polynomial-time functionals is the same class as $\mathbf{BFF}_2$. In [Set94] Seth generalized Definition 3 as follows.

Definition 4 ([Set94]) *Let $H : (\mathbf{N} \to \mathbf{N}) \times \mathbf{N} \to \mathbf{N}$ be a computable functional. Define $DTIME(H)$ to be the class of type-2 functionals such that, for every functional $F \in DTIME(H)$, there is an OTM that computes F and, on every $(f, x) \in (\mathbf{N} \to \mathbf{N}) \times \mathbf{N}$, the OTM halts within $H(f, x)$ steps.* ■

Clearly, Definition 4 is a direct lift of the typical notion of type-1 complexity classes, where the time bound is lifted to the type-2 functional. Note that the length function can be implicitly computed by the time bound, H, but such computation is certainly not feasible. Seth in [Set94] speculated that Definition 4 may lead to a general type-2 complexity theory with some type-2 complexity results analogous to the classical type-1 Gap, Union, or Speedup Theorems.

However, a type-2 functional seems too powerful for clocking purposes and the length function is difficult to compute. Another approach was also proposed by Seth in [Set94]. In fact, it is a generalized idea of Cook's POTM [Coo91].

Definition 5 ([Set94]) *Let $t : \mathbf{N} \to \mathbf{N}$ be recursive. Define $DTIME(t)$ to be the class of type-2 functionals such that, for every functional $F \in DTIME(t)$, there is* *an OTM that computes F and, on every $(f, x) \in (\mathbf{N} \to \mathbf{N}) \times \mathbf{N}$, the OTM halts within $t(m)$ steps, where $m = |\mathsf{max}(\{x, Q\})|$ and Q is the set of all answers returned from the oracle (f) during the course of the computation.* ■

A time bound is the limit we set up for our concerned computation, so that the execution time of the computation cannot exceed the bound. Thus, for practical reasons, Definition 5 is certainly not reasonable, because we have to complete our computation in order to obtain Q. In other words, we are not able to compute the time bound until Q is obtained.[21] Moreover, the main difficulty of Definition 5 is that Q may not be computable at all. Thus, it is impossible to have a clocking scheme based on Definition 5. To fix this problem, we can find a compromised idea in the definition of POTM's.

Definition 6 (POTM's [CK89, Coo91]) *Let M be an OTM. We say that M is a polynomial OTM (POTM for short) if there is a polynomial p such that, on every input at any step t during the course of the computation, the cost of the computation through step t is bounded by $p(|\mathsf{max}(Q_t, x)|)$, where $x \in \mathbf{N}$ is the type-0 input and Q_t is the list of answers returned from the oracle (f) during the period from the beginning of the computation up to step t.* ■

The definition of POTM's gives a much clearer idea of how to clock an OTM. In [Set94] Seth modified the idea involved in Definition 6 and gave a clocking scheme to solves the difficulties we just mentioned. Under his scheme, the clock is set based on the type-0 input at the beginning of the computation. And, during the computation, we dynamically update the clock based on answers returned from the oracle. Also, we keep tracking the number of steps the clocked OTM performs. Whenever we find that the number of steps the clocked OTM performs exceeds the time bound provided by

[21] In Definition 4, the time bound is computed by another type-2 functional.

the clock, we terminate the computation. The clock does not ask for information from the type-1 object but passively updates the time bound according to the computation being clocked. In such a way, the clock essentially can be computed as a type-1 function, since the number of queries made at any moment during the computation must be finite. A detail description of such clocking scheme for OTMs can be found in [Roy97, IKR01a].

However, there is an problem. Using arbitrary recursive type-1 clocks in this clocking scheme does not meet a basic requirement that appears in the original type-1 clocking scheme, namely that every clocked machine must be total. The above OTM clocking scheme cannot guarantee this. Even if the type-1 clock does have some nice properties such as being polynomial-time computable, the clocked OTM still may run forever on some inputs. We will demonstrate a POTM that is not total in the proof of Theorem 38 on page 66. In order to fix this problem, we shall impose more constraints on the clocks for our type-2 clocking scheme. These will be discussed in detail in Chapter 3, where we give some properties on our type-2 time bounds ($\mathbf{T_2TB}$) that are needed for our purposes.

1.4 Oracle Turing Machines: OTMs

As our model of type-1 computation, we use standard Turing Machines (TMs) with the conventions stated in Section 1.4.1. For type-2 computation we use Oracle Turing Machines (OTMs) as our standard computational model stated in Section 1.4.2. The time cost of a TM or OTM computation is simply the number of steps performed in the computation. Time cost will be the primary complexity measure studied here. For OTMs there are two standard cost models, which differ on the cost of oracle queries. As we will see, these two models can lead to very different complexity results. We shall demonstrate this difference in Section 1.4.2.

1.4.1 Turing Machines

Our standard model of type-1 computation is the deterministic, multi-tape Turing Machine (TM) as presented in [HU79]. All of our TMs have tape-alphabet $\{\mathbf{0}, \mathbf{1}, \mathbf{B}\}$, where $\mathbf{B}$ denotes the blank symbol. All tapes will be initialized to blank. Inputs and outputs of our TMs will be natural numbers in binary notation as strings of **0**'s and **1**'s. Thus, every $x \in \mathbf{N}$ will be encoded into a **0**-**1**-string of length $|x| = \lceil \log_2 x \rceil$ whenever I/O is needed. Moreover, we require our TMs to scan the entire input string at the beginning of the computation.

Let $\mathcal{PR}$ denote the set of partial recursive functions and $\mathcal{R}$ the set of recursive functions. Let $\langle \varphi_i \rangle_{i \in \mathbf{N}}$ be a fixed *acceptable programming system* for $\mathcal{PR}$ based on an indexing of our TMs. (See [Rog92] and [RC94] for more details on acceptable programming systems.) Thus, we say that TM M_e is coded by the φ-program e. Let $\langle \Phi_i \rangle_{i \in \mathbf{N}}$ denote a complexity measure associated with $\langle \varphi_i \rangle_{i \in \mathbf{N}}$. Unless stated otherwise, $\Phi_e(x)$ simply means the number of steps M_e performs on input x. (See [Blu67] or Section 5.1 for details on complexity measures.)

1.4.2 Oracle Turing Machines

The Oracle Turing Machine OTM was introduced by Turing himself for the study of relative computability [Dav58, Rog92, Soa87, Odi89]. Our OTMs are TMs with the addition of two special one-way-infinite tapes (i.e., a *query tape* and an *answer tape*) and a special query state for querying an oracle (see Figure 1.4.2).

An OTM on input $x \in \mathbf{N}$ with oracle $f : \mathbf{N} \to \mathbf{N}$ behaves like an ordinary TM except that on occasion it can write down a $q \in \mathbf{N}$ on its query tape and go into the

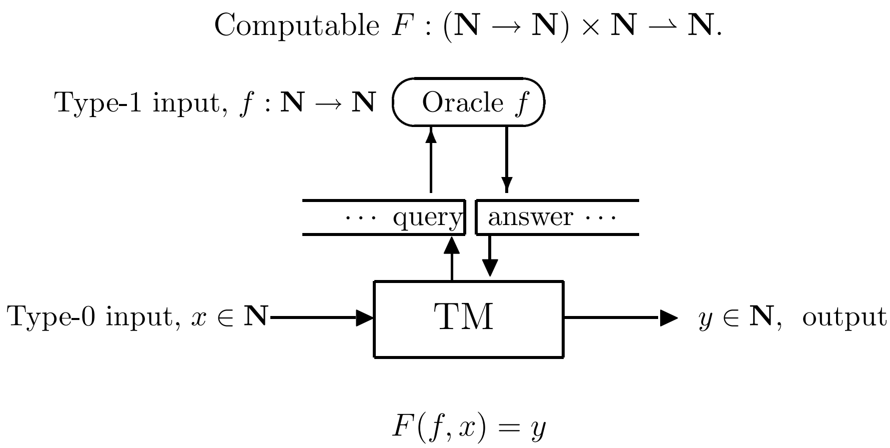

Figure 1: An Oracle Turing Machine (OTM)

query state. At that point, the oracle is viewed as taking control with the following actions:

1. It erases all non-blank symbols on the answer tape and returns that tape's head to the beginning of the tape.

2. It writes $f(q)$ on the answer tape.

3. It erases all non-blank symbols on the query tape and returns that tape's head to the beginning of the tape.

4. It then returns control to the OTM.

After taking back the control, the OTM is in the state following its query state. Thus, an OTM with oracle $f : \mathbf{N} \to \mathbf{N}$ is simply a TM that we have augmented with the ability to determine the value of f at any given value $q \in \mathbf{N}$. The four actions above all together cost the OTM one step.

The above describes OTMs with function oracles. It is more usual in the literature to see OTMs with set oracles, which are OTMs with an oracle $A \subseteq \mathbf{N}$ that can ask oracle questions of the form "$q \in A$?". However, by replacing A with its characteristic function, we see that the set-oracle model is simply a special case of the function-oracle model.

We restrict our standard model to OTMs with one total-function oracle. Thus, an OTM determines a type-2 functional of type $(\mathbf{N} \to \mathbf{N}) \times \mathbf{N} \rightharpoonup \mathbf{N}$, where the two arguments are the function oracle and the numerical input.

Definition 7

1. *Suppose that $\widehat{M}$ is an OTM, $f : \mathbf{N} \to \mathbf{N}$, and $x \in \mathbf{N}$. $\widehat{M}(f, x)$ denotes the output value, if any, of $\widehat{M}$ on input x with oracle f. $\widehat{M}(f, x)\uparrow$ means that $\widehat{M}(f, x)$ is undefined, which is the case when $\widehat{M}$ on input x with oracle f fails to halt. $\widehat{M}(f, x)\downarrow$ means $\widehat{M}(f, x)$ is defined and $\widehat{M}(f, x)\downarrow= z$ means $\widehat{M}(f, x)$ is defined with output z.*

2. *A functional $F : (\mathbf{N} \to \mathbf{N}) \times \mathbf{N} \rightharpoonup \mathbf{N}$ is **computable** iff there is an OTM that determines F. That is, there is an OTM, $\widehat{M}$, such that, for every $(f, x) \in (\mathbf{N} \to \mathbf{N}) \times \mathbf{N}$, we have*

$$F(f, x) = \begin{cases} \widehat{M}(f, x) & \textit{if} \quad \widehat{M}(f, x)\downarrow; \\ \uparrow & \textit{if} \quad \widehat{M}(f, x)\uparrow . \end{cases}$$

*We say such an $\widehat{M}$ **computes** F.* ■

We also fix an acceptable programming system $\langle\widehat{\varphi}_i\rangle_{i\in\mathbf{N}}$ for the computable functionals of type $(\mathbf{N} \to \mathbf{N}) \times \mathbf{N} \rightharpoonup \mathbf{N}$ based on our OTMs.[22] Thus, we say that $\widehat{\varphi}_e$ is

[22]See [Rog92] and [RC94] for details on relativized acceptable numbering. We shall formally state the relativized s-m-n and recursion theorems for $\langle\widehat{\varphi}_i\rangle_{i\in\mathbf{N}}$ in Section 1.5.

the functional computed by the $\widehat{\varphi}$-program e. Specially, we fix an indexing of OTMs, $\langle \widehat{M}_i \rangle_{i \in \mathbf{N}}$, and, for each $e \in \mathbf{N}$, $\widehat{\varphi}_e$ is the functional computed by the OTM, $\widehat{M}_e$.

In order to pindown some complexity issues, we consider two conventions on how OTMs treat answers returned from oracles; we also consider cost models for OTM computations associated with these two conventions. The two models lead to two very different complexity theories.

1. **Answer-Length-Cost Model:**

   Under this model, right after the oracle answer is returned, the OTM must scan the entire answer string until a $\mathbf{B}$ is encountered. Thus, any oracle query "$f(q) =?$" costs a total of $|q|+|f(q)|+1$ steps. In other words, we always charge the OTM for any oracle answer with the cost of its length, even if the answer is not subsequently used. We call this model the *Answer-Length-Cost Model*, and use OTM_A as an abbreviation of "OTM under the Answer-Length-Cost Model."

2. **Unit-Cost Model:**

   Under this model, the OTM does not have to read the oracle answer until needed. Moreover, the OTM is free to read only a portion of the entire oracle answer. Each oracle query "$f(q) =?$" therefore may cost as few as a total of $|q|$ steps. The size of the oracle answer is of no importance. It will be scanned upon demand during the computation and the expenses will be counted then. For example, reading the least significant bit only is enough to decide if the oracle answer is odd or even. We call this model the *Unit-Cost Model*, and use OTM_U as an abbreviation of "OTM under the Unit-Cost Model."

We take the Answer-Length-Cost model as our primary type-2 computational model. Thus, unless explicitly stated in a given context that $\widehat{M}_e$ can stand for either model, we let $\widehat{M_e}$ refer to an OTM_A. We use $\widehat{\Phi}_e(f, x)$ to denote the the number of steps $\widehat{M_e}$ performs on (f, x) under the answer-length-cost model. We use $\widehat{M_e^u}$ to refer to an OTM_U, and $\widehat{\Phi}_e^u(f, x)$ to denote the number of steps $\widehat{M_e^u}$ performs on (f, x). It is clear that both $\widehat{\Phi}$ and $\widehat{\Phi}^u$ satisfy Blum's two axioms: (i) $\widehat{\Phi}_e(f, a)\downarrow \Leftrightarrow\ \widehat{\varphi}_e(f, a)\downarrow$, and (ii) $\widehat{\Phi}_e(f, a) = y$ is recursively decidable (and likewise with $\widehat{\Phi}^u$).

The following example shows the computational difference between the answer-length-cost model and the unit-cost model.

Example 8 *Define* $F : (\mathbf{N} \to \mathbf{N}) \times \mathbf{N} \to \mathbf{N}$ *by:*

$$F(f, x) = \begin{cases} 0 & \textit{if } f(x) \textit{ is even;} \\ 1 & \textit{if } f(x) \textit{ is odd.} \end{cases} \tag{2}$$

The fastest $\widehat{M_e}$ for computing F on $(f, a) \in (\mathbf{N} \to \mathbf{N}) \times \mathbf{N}$ has $\widehat{\Phi}_e(f, a) \in \Theta(|a| + |f(a)|)$ in asymptotic Θ-notation [CLR89]. On the other hand, for the best $\widehat{M_e^u}$, we have $\widehat{\Phi}_e^u(f, a) \in \Theta(|a|)$.

We consider OTM_U too arbitrary in its use of the power of the oracle. We will see the structure of the complexity theory based on OTM_U has many problematic features.

1.5 Type-2 S-m-n and Recursion Theorems

Notations: Let $\mathcal{T}$ denote the set of total functions, i.e., $\mathbf{N} \to \mathbf{N}$. Recall that $\langle \widehat{\varphi}_i \rangle_{i \in \mathbf{N}}$ is an acceptable programming system for computable type-2 functionals based on our OTMs. Let $\langle \cdot, \cdot \rangle : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ be a fixed pairing function (see [Rog92], page 64), which is simply a computable bijection between $\mathbf{N} \times \mathbf{N}$ and $\mathbf{N}$.

The s-m-n theorem and the recursion theorem are essential tools in the study of computable functions. After Kleene's [Kle59, Kle63], there have been many approaches to extend recursion theory to higher-type functionals.[23] Following Kleene's notations and his S1 - S9 of [Kle59] (the inductive definition for the notion of higher-type computability), we can establish an s-m-n theorem for higher-type *countable* functionals in the following from:

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

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

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

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

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

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

Theorem 9 (Type-2 s-m-n theorem on type-0 argument) *There exists a recursive function $s : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ such that, for every $f \in \mathcal{T}$ and $e, x, y \in \mathbf{N}$, we*

[23] In Odifreddi's [Odi89], page 199, or Shoenfield's [Sho95], one can find a brief discussion about Kleene's work on the subject. For more details, see Normann's [Nor80] or Gandy and Hyland's [GH77].

have

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

■

Theorem 10 (Type-2 recursion theorem) *There is a recursive function $r : \mathbf{N} \to \mathbf{N}$ such that, for every $f \in \mathcal{T}$ and $e, x \in \mathbf{N}$, we have*

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

■

The two theorems above are key tools in what follows.

1.6 Our Goals and Achievements

Blum's [Blu67] axiomatic approach to formalize type-1 complexity is a tempting model to follow, especially because of its simplicity and generality. However, general type-2 complexity theory is a largely unmapped territory, and it is very difficult to construct a sensible and general theory without the help of a collection of some worked out special cases. Thus, instead of following Blum's approach, we adopt Hartmanis and Stearns' [HS65] approach of studying a very specific machine model and cost measure and examining key complexity-theoretic questions in this context. Our model of computation is the OTM model introduced in Section 1.4, and our cost measure is simply the number of steps of an OTM computation. Our research begins with the search for answers to the key problems proposed at the beginning of this chapter:

1. What are appropriate resource-bound functions for type-2 computation?

2. What is the size of a type-2 problem?

3. What is a proper concept of type-2 "almost everywhere" relations?

1.6.1 Goals

The inherent computational complexity of function $f : \mathbf{N} \to \mathbf{N}$ is fundamentally an asymptotic notion. This is because, in any reasonable model of computation, given a program for f and a finite set $S \subset \mathbf{N}$, one can patch this program so that for $x \in S$ the cost of the computation is essentially the cost reading the input and writing the output, and for every $x \notin S$ the cost of the computation is essentially what it was with the original program. Therefore, in general, to show that a certain $f : \mathbf{N} \to \mathbf{N}$ is not computable within resource bound $t : \mathbf{N} \to \mathbf{N}$, we must show that any program for f must infinitely often exceed the resource bound t. Because of this asymptotic nature of complexity, it is usual to define the complexity class $DTIME(t)$ as the collection of functions $f : \mathbf{N} \to \mathbf{N}$ such that, for each such function f in the class, there is some program for f that runs within time bound $t(|x|)$ for all but finitely many $x \in \mathbf{N}$. In other words, we say that $f \in DTIME(t)$ if and only if there is a program for f with a cost function less than or equal to the resource bound t "almost everywhere." Many of the fundamental results in type-1 complexity theory have proofs that make essential use of the looseness afforded by the "for all but finitely many" clause in the definition of complexity class. Seth [Set94] suggests lifting this definition of complexity class into type-2 and projects that this may lead to some key type-2 complexity theorems analogous to those in classical complexity theory. In order to do this, we need to have a formal definition of type-2 complexity classes along the lines of the standard approach for type-1. Thus, the first major goal of this thesis is:

1. To define an appropriate type-2 version of the "almost everywhere less than or equal to" relation.

As we will see, the direct lift of a type-1 bounding function quickly runs into trouble. Thus, our next major goal is:

2. To define a class of functions that are suitable to serve as time bounds for type-2 computation.

Once we have met our first two goals, we can then move to our third goal:

3. To define type-2 complexity classes based on our notion of a type-2 almost-everywhere relation and time bounds for type-2 computation.

Once we have the definitions and supporting notions in hand, we move on to our final goal which is:

4. To apply our notion of type-2 complexity classes to obtain type-2 versions of several key theorems in complexity theory.

In part the results in our fourth goal serve to validate our prior definition, but these results also provide the first glimpse at the structure of general type-2 complexity theory.

1.6.2 Summary of Our Work

Chapter 2

Type-2 Almost-Everywhere Relations: We define a type-2 almost-everywhere relation $\leq_{\mathbf{2}}^{*}$ such that, for any type-2 functionals F and G, $F \leq_{\mathbf{2}}^{*} G$ means that there are only a "small amount" of inputs on which $F \leq G$ does not hold. In order to give a precise notion of "small amount," we study the close relation between topologies on the space $(\mathbf{N} \to \mathbf{N}) \times \mathbf{N}$ and type-2 computation under our OTM model.

Our type-2 almost-everywhere relation $\leq_{\mathbf{2}}^{*}$ must satisfy some key intuitive requirements. For example, $\leq_{\mathbf{2}}^{*}$ must be transitive and, for computable F and G, $F \leq_{\mathbf{2}}^{*} G$ means that there are only finitely many computations to realize that $F \leq G$ does not hold. The compactness of a set of type-1 objects seems to be the right notion of a "small amount" in that context. In the study of type-2 computation, there are two widely used topologies for the space $(\mathbf{N} \to \mathbf{N}) \times \mathbf{N}$, the *positive information topology* [Odi89] and the *Baire topology* [Rog92, Odi89, AGM92]. However, these two topologies turn out to be inappropriate for our purposes, as we will explain. What turns out to work is to define $\leq_{\mathbf{2}}^{*}$ via a collection of induced topologies.

Chapter 3

Type-2 Time Bounds: In the light of Seth's [Set94] clocking scheme, we define a class of functions, Type-2 Time Bounds ($\mathbf{T}_2\mathbf{TB}$), of type $\mathcal{F} \times \mathbf{N} \to \mathbf{N}$, where $\mathcal{F}$ denotes the set of finite functions over $\mathbf{N}$. We show that every $\beta \in \mathbf{T}_2\mathbf{TB}$ is suitable for clocking our OTMs. In particular, we require that such a β has to be effectively computable and convergent. Convergence means that, if we keep feeding information of some fixed total function to a $\beta \in \mathbf{T}_2\mathbf{TB}$, β will eventually refuse to change its value again. In such a way, all clocked OTMs are total and the problem of having non-total POTMs is solved.

If we can effectively decide whether a given β has converged, we call such β locking detectable. We study this property and show that not every type-2 time bound is locking detectable. We also study the relation between our type-2 time bounds and the limit functionals formalized by Rogers [Rog92].

Clocked Oracle Turing Machines: We provide a standard clocking scheme for both OTM cost models: the answer-length-cost model and the unit-cost model. Every $\beta \in \mathbf{T_2TB}$ can be used as a clock in the scheme to monitor the cost of the computation. The scheme updates the clock dynamically during the course of the computation. If the cost exceeds the time bound, the computation will be terminated immediately. Since at any point in the course of a computation only a finite portion of the type-1 input can contribute to the computation, the value of the clock should not be calculated based on irrelevant information to the computation being clocked. We show that $\mathbf{T_2TB}$ is broad enough so that, for every total computable functional, there is a β-clocked OTM that computes it. Moreover, every OTM $\widehat{M}_e$ clocked by any $\beta \in \mathbf{T_2TB}$ computes a total type-2 functional $\widehat{\varphi}_{e,\beta} : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$.

Chapter 4

Type-2 Complexity Classes: Based on the ideas and tools developed in Chapters 2 and 3, we define a type-2 complexity class $\mathbf{C}(\beta)$ for each $\beta \in \mathbf{T_2TB}$. We show that each such complexity class $\mathbf{C}(\beta)$ has some nice properties. For example, $\mathbf{C}(\beta)$ is recursively representable under some reasonable conditions on β, and hence there is a *programming system* [RC94, Rog92] for such $\mathbf{C}(\beta)$. This also implies that there is a programming system for some union of type-2 complexity classes (e.g., $\mathbf{BFF_2}$). Moreover, $\mathbf{C}(\beta)$ has the following intuitive property: if $F \in \mathbf{C}(\beta)$, then the cost of computing F is bounded by F_β almost everywhere, where F_β is the limit functional determined by β.

We point out that a $\beta \in \mathbf{T_2TB}$ can determine two different type-2 complexity classes under the two different OTM cost models. The differences turn out to be essential in many theorems. Thus, the two OTM cost models lead to two different type-2 complexity theories.

Equivalent Type-2 Time Bounds: Let $\alpha, \beta \in \mathbf{T_2TB}$. It is possible that $\alpha \neq \beta$, but $\mathbf{C}(\alpha) = \mathbf{C}(\beta)$. If this is the case, we say that α and β are equivalent. We are especially interested in the equivalence between locking detectable and non-locking detectable type-2 time bounds. We show that, for every non-locking detectable type-2 time bound, we can uniformly construct an equivalent one that is locking detectable. Thus, we can assume the type-2 time bounds to be locking detectable without loss of generality. This assumption simplifies many proofs.

Chapter 5

In this chapter we briefly discuss Blum's axiomatic formulation of complexity measures [Blu67]. We then describe, without proofs, some important complexity theorems in type-1 complexity theory, including the Union Theorem [MM69], the Gap Theorem [Bor69, Bor72], the Operator Gap Theorem [Con72, Con73], the Compression Theorem [Blu67], the Honesty Theorem [MM69], the Speedup Theorem [Blu67], and the Operator Speed-up Theorem [MF72]. We point out the significance of these classic theorems and their relations. We identify type-2 analogs of these results as our target theorems to be proved or disproved in the following two chapters.

Chapter 6

Type-2 Union Theorems: We prove that, in general, the class of two type-2 complexity classes is not closed under union. Moreover, a direct lift of the condition for the type-1 Union Theorem alone does not suffice to prove a Union Theorem for type-2. Thus, we provide a reasonable strong condition for our type-2 Union Theorem under the answer-length-cost model. However, this condition is still not sufficient to prove the Union Theorem under the unit-cost model.

Under the answer-length-cost model, we define a type-2 analog of the big-O notation, $\mathbf{O}(\beta)$, for $\beta \in \mathbf{T}_2\mathbf{TB}$. The definition suggests that we can ignore the constant term and factor in $c\beta + d$ for $c, d \in \mathbf{N}$. In other words, for every $c, d \in \mathbf{N}$, we have $\mathbf{O}(c\beta + d) = \mathbf{O}(\beta)$. Our Union Theorem under the answer-length-cost model asserts that $\mathbf{O}(\beta)$ is a type-2 complexity class. That is, there is an $\alpha \in \mathbf{T}_2\mathbf{TB}$ such that $\mathbf{C}(\alpha) = \mathbf{O}(\beta)$. Also, we prove that if $\alpha, \beta \in \mathbf{T}_2\mathbf{TB}$ and $\alpha \leq^* \beta$, then $\mathbf{O}(\alpha + \beta) = \mathbf{O}(\beta)$. These theorems justify our definition of $\mathbf{O}(\beta)$.

A Type-2 Gap Theorem: Under the answer-length-cost model, we use an easy trick to lift the gap phenomenon from type-1 and prove that, for any recursive function $g : \mathbf{N} \to \mathbf{N}$, there is a $\beta \in \mathbf{T}_2\mathbf{TB}$ such that $\mathbf{C}(\beta) = \mathbf{C}(g \circ \beta)$. However, if we require the β constructed for the Gap Theorem to meet some nontrivial conditions, then the trick does not work under the unit-cost model. We call this opposite theorem the Anti-Gap Theorem, which asserts that, if the recursive function g is sufficiently large, then for every $\beta \in \mathbf{T}_2\mathbf{TB}$ under a rather weak condition, we have $\mathbf{C}(\beta) \subset \mathbf{C}(g \circ \beta)$ under the unit-cost model.

A Type-2 Compression Theorem: If we replace the gap factor g in the Gap Theorem by a recursive operator $\Theta : \mathbf{T}_2\mathbf{TB} \to \mathbf{T}_2\mathbf{TB}$, we have a theorem opposite to the original Operator Gap Theorem. The theorem asserts that there is a recursive operator Θ such that, for every $\beta \in \mathbf{T}_2\mathbf{TB}$, we have $\Theta(\beta) \in \mathbf{T}_2\mathbf{TB}$ and $\mathbf{C}(\beta) \subset \mathbf{C}(\Theta(\beta))$. We call this theorem the Inflation Theorem, because we can uniformly inflate any type-2 complexity class.

Type-2 Honesty: In type-1, the original purpose of the Honesty Theorem is to show that the restriction[24] of the Compression Theorem does not omit any complexity class. In other words, the gap phenomenon can be removed without loss of generality. Since our type-2 time bounds are represented by type-1 functions, we can require a $\beta \in \mathbf{T_2TB}$ to be honest as an ordinary type-1 recursive function. Also, we can embed $\mathbf{T_2TB}$ in a measured set. In fact, we can directly lift the original proof of the Honesty Theorem into type-2. However, we do not think such work is necessary because of the Inflation Theorem by which we don't have to embed our $\mathbf{T_2TB}$ in a measured set to remove the gap phenomenon.

Chapter 7

A Type-2 Speed-up Theorem: The Speed-up Theorem asserts that there is a total computable function that fails to have an optimal program for it. That is, there is a recursive function f such that, given any program for f, there is always another program for f that runs faster almost everywhere. We solve some difficulties caused by the oracle queries and prove the Speed-up Theorem in our type-2 programming system. In addition, we prove the type-2 Operator Anti-Speed-up Theorem that is opposite to the type-2 analog of Operator Speed-up Theorem.

Type-2 Query Speed-up Theorems: We consider the number of oracle queries made throughout the course of a type-2 computation as a special dynamic computational resource in type-2. Thus, we study the query complexity and give three alternatives for defining a *query-optimum* program: *absolute, strong,* and *weak* query-optimum programs. We show that there is a computable functional without an absolute query-optimum program nor a strong query-optimum program for it. This means

[24]The original Compression Theorem restricts the functions that determine complexity classes to a *measured set* [Blu67]. Note that, functions in a *measured set* are honest.

that we have speed-up theorems associated with the absolute and strong concepts of query-optimum programs. However, we do not know if there is a speedable program in terms of weak query-optimum.

Chapter 8

This chapter concludes the dissertation by pointing out some possible directions for future study towards a general type-2 complexity theory. We also list some possible applications of our results such as the complexity issue in machine learning and programming languages.

Chapter 2

Topology and Type-2 Computation

In this chapter we study the close relationship between topologies and type-2 computation. By using the idea of compactness from topology, we define a notion of "small sets" of type-1 objects and, based on that notion, we establish an "almost-everywhere" relation $\leq_2^*$ on type-2 objects. The notion of compact sets turns out to be an essential tool for defining type-2 complexity classes in the next two chapters. We will motivate our definition of $\leq_2^*$ by considering some alternative topologies for the notion of "small sets" of type-1 objects and hence of type-2 almost-everywhere relations. Those topologies seem reasonable at first, but prove unsatisfactory on closer inspection.

Notations and Conventions

Let $\mathcal{P} \triangleq \mathbf{N} \rightharpoonup \mathbf{N}$ be the set of partial functions over $\mathbf{N}$, and let $\mathcal{F}$ denote the set of finite functions over $\mathbf{N}$. Recall our standard pairing function $\langle \cdot, \cdot \rangle$, which is a bijection between $\mathbf{N} \times \mathbf{N}$ and $\mathbf{N}$. A set $S \subset \mathbf{N}$ is said to be *single-valued* if and only if, for all $x \in \mathbf{N}$, there exists at most one $y \in \mathbf{N}$ such that, $\langle x, y \rangle \in S$. Clearly, for each element σ in $\mathcal{F}$, there exists a unique $S \subset \mathbf{N}$ which is finite, single-valued, and, for every $(x, y) \in \mathbf{N} \times \mathbf{N}$,

$$[\sigma(x) = y] \iff [\langle x, y \rangle \in S].$$

Thus, we can identify each element in $\mathcal{F}$ by the canonical index of its corresponding single-valued set. (See [Rog92], page 70, for more details.) Unless stated otherwise, lowercase Greek letters σ and τ range over $\mathcal{F}$. Let $\mathsf{dom}(\sigma)$ denote the domain of σ and let $\mathsf{card}(\sigma)$ denote the cardinality of $\mathsf{dom}(\sigma)$. For $\alpha, \beta \in \mathcal{P}$, we write $\alpha \subseteq \beta$ iff we have that, as sets of ordered pairs, $\alpha \subseteq \beta$. When $\alpha \subseteq \beta$, we say that β is an extension of α. By convention, let $\emptyset$ denote the everywhere undefined function. For each $\sigma \in \mathcal{F}$, let $\sigma^{\sim}$ denote the set of all extensions of σ, i.e.,

$$\sigma^{\sim} = \left\{ f \;\middle|\; f \in \mathcal{P} \ \wedge\ \sigma \subseteq f \right\}. \tag{4}$$

For $F : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$ and $(f, x) \in \mathcal{T} \times \mathbf{N}$, $F(f, x) \uparrow$ means that F is undefined on (f, x) and $F(f, x) \downarrow= y$ means that F is defined on (f, x) and its value is y. The same notation is also applied to type-1 functions. Moreover, $F^{-1}(A)$ denotes the inverse image of A, where $A \subseteq \mathbf{N}$.

2.1 Topology and Type-2 Continuous Functionals

Consider the following collection of subsets of $\mathcal{P}$,

$$\mathcal{B} = \left\{ \sigma^{\sim} \;\middle|\; \sigma \in \mathcal{F} \right\} \cup \left\{ \emptyset \right\}. \tag{5}$$

We observe that, for any $\sigma, \tau \in \mathcal{F}$,

$$\sigma^{\sim} \cap \tau^{\sim} = \begin{cases} \emptyset & \text{if } \exists a \in \mathsf{dom}(\sigma) \cap \mathsf{dom}(\tau)\ [\sigma(a) \neq \tau(a)]; \\ (\sigma \cup \tau)^{\sim} & \text{otherwise.} \end{cases} \tag{6}$$

Thus, $\mathcal{B}$ is closed under finite intersection. So $\mathcal{B}$ can be taken as a basis of the familiar *positive information topology* on the space $\mathcal{P}$ [Odi89]. Let this topology be denoted as $\mathbb{P}$. Let $\mathbb{P} \times \mathbb{N}$ denote the product topology from $\mathbb{P}$ and $\mathbb{N}$, where $\mathbb{N}$ is the discrete topology on $\mathbf{N}$.

Definition 11 *We say that $F : \mathcal{P} \times \mathbf{N} \rightharpoonup \mathbf{N}$ is* ***continuous*** *if and only if for any open set $A \in \mathbb{N}$, $F^{-1}(A)$ is open in $\mathbb{P} \times \mathbb{N}$.* ■

Since the finite subsets of $\mathbf{N}$ are a basis of $\mathbb{N}$, it follows that any finite set $A \subset \mathbf{N}$ is open in $\mathbb{N}$. Given any $F : \mathcal{P} \times \mathbf{N} \rightharpoonup \mathbf{N}$, we have that F is continuous if and only if for any finite set $A \subseteq \mathbf{N}$, $F^{-1}(A)$ is open in $\mathbb{P} \times \mathbb{N}$. Thus, when F is continuous, the inverse image of any finite set $A \subset \mathbf{N}$ must be able to be described by a union of basic open sets of $\mathbb{P} \times \mathbb{N}$.

Definition 12 *Let $F : \mathcal{P} \times \mathbf{N} \rightharpoonup \mathbf{N}$. We say that,*

1. *F is* ***compact*** *if and only if*

$$\forall (f, x) \in \mathcal{P} \times \mathbf{N}\ \exists \sigma \in \mathcal{F}\ \Big[F(f, x) = F(\sigma, x) \Big].$$

2. *F is* ***monotone*** *if and only if*

$$\forall (\sigma, x) \in \mathcal{F} \times \mathbf{N}\ \Big[F(\sigma, x)\downarrow\ \implies \forall \tau \supseteq \sigma (F(\tau, x)\downarrow = F(\sigma, x)) \Big].$$

■

The next two theorems are due to Uspenskii and Nerode ([Odi89], page 187).

Theorem 13 (Uspenskii [Usp55], Nerode [Ner57]) *A functional $F : \mathcal{P} \times \mathbf{N} \rightharpoonup \mathbf{N}$ is continuous if and only if F is compact and monotone.* ■

Recall from Definition 7 a functional $F : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$ is computable iff it can be computed by an OTM. If we restrict the type-1 input of functionals to be total, we can straightforwardly apply Uspenskii and Nerode's theorem to show:

Theorem 14 *Every computable functional of type $\mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$ is continuous.* ■

Note that the converse of the above theorem does not hold, i.e., not every continuous functional of type $\mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$ is computable. We wish to interpret Theorem 14 in terms of machines. The key to this interpretation is the simple observation that every terminating computation is finite. Let OTM $\widehat{M_e}$ compute $F : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$. Suppose that $\widehat{M_e}$ halts on $(f, a) \in \mathcal{T} \times \mathbf{N}$, i.e., $F(f, a) \downarrow$. Since every terminating computation must be finite, only a finite part of f is queried during the computation. Let this finite portion of f be σ. Then, any information about f outside σ has no effect on the computation. In other words, for any function $g \supset \sigma$, the computation of $\widehat{M_e}$ on (g, a) is the same as on (f, a), and thus $\widehat{M_e}(f, a) = \widehat{M_e}(g, a)$, and so $F(f, a) = F(g, a)$. We say that such a σ is a *locking fragment* with respect to the computation of $\widehat{M_e}$ on (f, a). Formally, we give the following definition.

Definition 15 *Let $F : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$ and $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$.*

1. *We say that (σ, a) is a **locking fragment** of F if and only if*

$$\exists y \in \mathbf{N}\ \forall g \supset \sigma \left[F(g, a) \downarrow = y \right].$$

2. *We say that (σ, a) is a **minimal locking fragment** of F if and only if (σ, a) is a locking fragment of F, and for every $\tau \in \mathcal{F}$ with $\tau \subset \sigma$, (τ, a) is not a locking fragment of F.* ■

Putting Theorems 13 and 14 and the above definitions together, it is clear to see that, if $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ is computable, then for every $(f, a) \in \mathcal{T} \times \mathbf{N}$, there must exist a unique $\sigma \in \mathcal{F}$ with $\sigma \subset f$ such that (σ, a) is a minimal locking fragment of F. However, such σ in general cannot be effectively decided.

2.1.1 The Topology $\mathbb{T} \times \mathbb{N}$ on $\mathcal{T} \times \mathbf{N}$

Since we restrict the oracles of our OTMs to total functions, it is more suitable to consider the topology $\mathbb{T}$, the Baire topology [Rog92, Odi89, AGM92], which is the topology induced on the Baire space $\mathcal{T}$ by $\mathbb{P}$. That is, the $\mathbb{T}$-open sets are precisely the sets of the form $\{f \mid f \in O \cap \mathcal{T}\}$, where O is some $\mathbb{P}$-open set.

Definition 16 *Let $\sigma \in \mathcal{F}$ and $a \in \mathbf{N}$. We define*

$$\begin{aligned} (\!(\sigma)\!) &= \left\{ f \;\middle|\; \sigma \subset f \ \wedge\ f \in \mathcal{T} \right\}, \\ (\!(\sigma, a)\!) &= \left\{ (f, a) \;\middle|\; \sigma \subset f \ \wedge\ f \in \mathcal{T} \right\}. \end{aligned}$$

■

Let $\mathbb{T} \times \mathbb{N}$ denote the product topology of $\mathbb{T}$ and $\mathbb{N}$.[1] For each $(\sigma, x) \in \mathcal{T} \times \mathbf{N}$, we take $(\!(\sigma, x)\!)$ as a basic open set of $\mathbb{T} \times \mathbb{N}$. Clearly, for every $\sigma_1, \sigma_2 \in \mathcal{F}$ and $x_1, x_2 \in \mathbf{N}$, we have

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

The intersection operation $(\!(\sigma_1)\!) \cap (\!(\sigma_2)\!)$ above simply follows the observation given in (6) on page 40. The union operation $(\!(\sigma_1, x_1)\!) \cup (\!(\sigma_2, x_2)\!)$ is conventional and may result in an open set that is not basic.[2]

[1]Note that $\mathcal{T} \cong \mathcal{T} \times \mathbf{N}$ via $f \mapsto (f', f(0))$, where $f'(x) = f(x+1)$. We make the distinction for the difference between the type-0 and type-1 inputs of the functionals we concerned.

[2]It is clear that, given any $f, g \in \mathcal{T}$ and $a \in \mathbf{N}$ such that $f \neq g$, there exist $\sigma \subset f, \tau \subset g$, and $k \in \mathbf{N}$ such that, $\sigma(k) \downarrow \neq \tau(k) \downarrow$. Therefore, $(f, a) \in (\!(\sigma, a)\!)$, $(g, a) \in (\!(\tau, a)\!)$, and $(\!(\sigma, a)\!) \cap (\!(\tau, a)\!) = \emptyset$, and hence $\mathbb{T} \times \mathbb{N}$ is a Hausdorff (T_2) topology on $\mathcal{T} \times \mathbf{N}$.

2.1.2 Compactness in $\mathbb{T} \times \mathbb{N}$

In this sub-section we shall discuss some topological properties of the topology $\mathbb{P} \times \mathbb{N}$ we just described. We first formally define some topological terminologies. (For more details on point set topology, see [Bau64].)

Definition 17 *Let $\mathbb{A}$ be a topology on space A and let $\mathcal{O}$ be a collection of subsets of A. Suppose $S \subseteq A$.*

1. *We say that $\mathcal{O}$ is a **cover** for S if and only if $S \subseteq \bigcup_{O \in \mathcal{O}} O$.*
2. *We say that $\mathcal{O}$ is an **open cover** for S if and only if (i) $\mathcal{O}$ is a cover for S and (ii) every set in $\mathcal{O}$ is $\mathbb{A}$-open, i.e., $\mathcal{O} \subseteq \mathbb{A}$.*
3. *If $\mathcal{S}$ and $\mathcal{O}$ are two covers for S and $\mathcal{S} \subseteq \mathcal{O}$, then $\mathcal{S}$ is said to be a **subcover** of $\mathcal{O}$ for S.*
4. *S is said to be **compact** if and only if for any open cover $\mathcal{O}$ for S, there exists a finite subcover of $\mathcal{O}$ for S.* ■

König's Lemma[3] (and its variant, the Fan Theorem) turn out to be useful in providing sufficient conditions for a set being compact in $\mathbb{T} \times \mathbb{N}$.

Terminology: A *branch* of a tree is a path from the root to a leaf. A tree T is said to be a *finitary tree* if T is a finitely branching tree: that is, every node of T has only finitely many immediate successors.

König's Lemma: *If T is a finitary tree with infinitely many nodes, then T has an infinite branch.*

[3]More details about König's Lemma can be found in [Odi89], page 505, and in [AGM92], page 753.

Fan Theorem: [4] *If all branches of the finitary tree T are finite, then T has a finite number of nodes.*

For $A \subset \mathbf{N}$, we say that A is *bounded* simply when A is finite. For $A \subset \mathcal{T}$, we say that A is *bounded* if there exists $g \in \mathcal{T}$ such that, for every $f \in A$, we have $f \leq g$ under the pointwise ordering. For $A \subset \mathcal{T} \times \mathbf{N}$, we say that A is *bounded* if there exists $g \in \mathcal{T}$ and $a \in \mathbf{N}$ such that $A \subseteq B_{g,a}$, where

$$B_{g,a} \;=\; \left\{ (f,x) \;\middle|\; f \leq g \text{ and } x \leq a \right\}.$$

The following two lemmas are useful tools to prove our next theorem.

Lemma 18 *Let $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be any continuous functional and $X \subset \mathcal{T} \times \mathbf{N}$. If X is compact in $\mathbb{T} \times \mathbb{N}$, then $F(X)$ is bounded.*

Proof: Suppose that $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ is continuous and $X \subset \mathcal{T} \times \mathbf{N}$ is compact. Since F is continuous, by Theorem 13 we know that, for every $(f,x) \in \mathcal{T} \times \mathbf{N}$, there exists $\sigma \subset f$ such that, (σ, x) is a minimal locking fragment of F. Thus, the set $\mathcal{O}$ in the following is well defined and is an open cover for X:

$$\mathcal{O} = \left\{ (\!(\sigma, a)\!) \;\middle|\; (\sigma, a) \text{ is a minimal locking fragment of } F \text{ on some } (f,a) \in X \right\}.$$

Since X is compact, there exists a finite sub-cover of $\mathcal{O}$ for X such as:

$$\mathcal{O}' = \left\{ (\!(\sigma_0, a_0)\!), (\!(\sigma_1, a_1)\!), \ldots, (\!(\sigma_n, a_n)\!) \right\},$$

for some $n \in \mathbf{N}$. Clearly, $F(\mathcal{O}')$ is finite because, for each $(\!(\sigma_i, a_i)\!) \in \mathcal{O}'$, (σ_i, a_i) is a locking fragment. Thus, $F(X)$ is also finite, and hence bounded. □

[4] The Fan Theorem is the contrapositive of König's Lemma. The Fan Theorem can be seen as a graphic version of the Bolzano-Weierstrass theorem, which states: any infinite subset of a bounded (closed) interval has an accumulation point in the interval. (See [Rog92], page 157.) Unlike König's Lemma, the Fan Theorem can be formulated into intuitionistic logic. More details on this issue can be found in [AGM92], page 754.

Lemma 19 *If $X \subset \mathcal{T}$ is unbounded, then there exists an $a \in \mathbf{N}$ such that the set $\{f(a) \mid f \in X\}$ is infinite.*

Proof: By contradiction, suppose that $X \subset \mathcal{T}$ is unbounded and for all $a \in \mathbf{N}$, $\{f(a) \mid f \in X\}$ is finite. If this is the case, we can define g by, for every $a \in \mathbf{N}$,

$$g(a) = \mathsf{max}\{f(a) \mid f \in X\}.$$

Clearly, $g \in \mathcal{T}$, and X is bounded by g, which contradicts our assumption that X is unbounded. $\square$

Theorem 20 *Let $X \subset \mathcal{T} \times \mathbf{N}$ and $X \neq \emptyset$.*

$$X \textit{ is } \textbf{\textit{compact}} \textit{ in } \mathbb{T} \times \mathbb{N} \iff X \textit{ is } \textbf{\textit{bounded}}.$$

Proof: Let $X \subset \mathcal{T} \times \mathbf{N}$ and $X \neq \emptyset$. Consider sets A and B defined by

$$A = \left\{ x \;\middle|\; (f, x) \in X \right\}, \tag{7}$$

$$B = \left\{ f \;\middle|\; (f, x) \in X \right\}. \tag{8}$$

Moreover, for any $i \in N$, define $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ by

$$\begin{aligned} F(f, x) &= x, \\ G_i(f, x) &= f(i). \end{aligned}$$

Clearly, both F and G_i are continuous.

For $\Rightarrow$ direction: Suppose that X is compact in $\mathbb{T} \times \mathbb{N}$. By Lemma 18, $F(X)$ is bounded, and hence A is bounded.

By way of contradiction, suppose that B is unbounded. Then, by Lemma 19, there exists $b \in \mathbf{N}$ such that the set $\{f(b) \mid f \in X\}$ is infinite. Fix such b

and consider G_b. Clearly, $G_b(X) = \{f(b) \mid f \in X\}$ is unbounded. But this contradicts the assertion of Lemma 18 that if G_b is continuous and X compact, then $G(X)$ must be bounded. Therefore, B is bounded, and hence X is bounded.

For $\Leftarrow$ direction: Suppose that X is bounded. Thus, A and B are bounded. Since A is finite, it follows that A is compact in $\mathbb{N}$.

By way of contradiction, suppose that B is not compact in $\mathbb{T}$ and $\mathcal{O}$ is an open cover without finite sub-cover for B. Without loss of generality, we may assume that

$$\mathcal{O} = \left\{ (\!(\sigma_i)\!) \;\middle|\; i \in \mathbf{N} \right\},$$

where each $\sigma_i \in \mathcal{F}$ is an initial fragment of some total function. Also, assume that, for every $i, j \in \mathbf{N}$ with $i \neq j$, $(\!(\sigma_i)\!) \cap B \neq \emptyset$ and $\sigma_i \neq \sigma_j$.

Let T be a tree as follows: (i) The root represents the empty set. (ii) Each branch of T represents exactly one σ_i, and each σ_i is represented by exactly one branch of T. (iii) For every $i \neq j$, $\tau = \sigma_i \cap \sigma_j$ is represented by a unique path from an internal node to the root of T.

Because of the assumption on $\mathcal{O}$ and that B is bounded, T must be a finitary tree with infinitely many nodes. By König's Lemma, T has an infinite branch. Since we assume that $(\!(\sigma_i)\!) \cap B \neq \emptyset$ for each $i \in \mathbf{N}$, it follows that we can find an $f \in B$ along the infinite branch which is not covered by any set in $\mathcal{O}$. This contradicts that $\mathcal{O}$ is an open cover for B. Thus, B must be compact. It follows that X is compact. $\square$

Here we examine the relation between a compact set in $\mathbb{T} \times \mathbb{N}$ and the computation for a total computable functional on it. Fix any computable $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$. Consider the bounded set $B \subset \mathcal{T} \times \mathbf{N}$ defined by

$$B = \left\{ (f, 0) \;\middle|\; f \text{ is 0-1 valued} \right\}.$$

It is clear that B is bounded, and hence by Theorem 20, B is compact. We claim that for any OTM $\widehat{M_e}$ that computes F, there are only finitely many computations of $\widehat{M_e}$ on B. This is because otherwise, by König's Lemma, there would be $(f, 0) \in B$ such that, $\widehat{M_e}$ on $(f, 0)$ will query f-oracle infinitely often, and hence $\widehat{M_e}$ will run forever. This contradicts the assumption that F is total and $\widehat{M_e}$ computes it. We generalize this observation into the following theorem with the proof omitted.

Theorem 21 *For every $\widehat{\varphi}_e : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ and every set $B \subset \mathcal{T} \times \mathbf{N}$, if B is compact in $\mathbb{T} \times \mathbb{N}$, then there are finitely many computations of $\widehat{M_e}$ on B.* ■

2.2 Type-2 Almost-Everywhere Relations

For $f, g \in \mathcal{T}$, we say that f is almost everywhere less than g (written as $f \leq^* g$) if and only if $f(x) < g(x)$ for all but finitely many $x \in \mathbf{N}$. The $\leq^*$ relation plays a key role in the development of type-1 complexity theory. It allows a "reasonable tolerance" of exception in defining a complexity class. In the following sections we investigate a few type-2 analogs of $\leq^*$ for our purposes in type-2 complexity.

For type-2 F and G, a straightforward analogue $F \leq^*_{\mathbf{2}} G$ would be:

$$\text{"For all but finitely many } (f, x),\ F(f, x) \leq G(f, x).\text{"} \tag{9}$$

However, (9) is too restrictive to be interesting because there is no finite computation that can positively realize that $f = g$ for the type-1 f and g. Thus, if we restrict F and G to computable functionals, a better notion of $F \leq^*_{\mathbf{2}} G$ should be something like:

"For all but finitely many computations we have $F \leq G$."

More precisely, we should say that, for computable $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$,

> F is almost everywhere less than G if and only if there exists two $\widehat{\varphi}$-programs i and j such that for F and G, respectively, such that, for all but finitely many computations of i and j, we have that the output of $\widehat{\varphi}$-program i is less than or equal to the putput og $\widehat{\varphi}$-program j. (10)

In the light of Theorem 21, it seems reasonable to formalize (10) as:

$$F \leq_{\mathbf{2}}^{*} G \iff F(f,x) \leq G(f,x), \quad \text{for all } (f,x) \in \mathcal{T} \times \mathbf{N} \text{ but a } (\mathbb{T} \times \mathbb{N})\text{-compact set.} \tag{11}$$

However, we shall see below that the two notions of "compactness" with respect to topology $\mathbb{T} \times \mathbb{N}$ and "finitely many computations" do not match up quite right for our purposes. Consequently, we will consider alternative topologies in which to formalize (11). We first state two goals in our mind.

2.2.1 Requirements

Here we summarize the discussion above and set up two goals for a proper type-2 relation $\leq_{\mathbf{2}}^{*}$ that seems reasonable to require.

Goal 1: Suppose that type-2 functionals F and G are computable. We want $F \leq_{\mathbf{2}}^{*} G$ to be the case that there are two OTMs that compute F and G, respectively, and there are only finitely many computations of the two OTMs resulting in values such that $F > G$.

Goal 2: The type-2 relation $\leq_2^*$ should be transitive. In type-1, transitivity is trivial but it preserves an important property for defining complexity classes. That is, we should not lose any function in a complexity class by increasing the resource bound. This requirement maintains a basic hierarchy among complexity classes. For the obvious reason, we want to preserve this hierarchy in defining type-2 complexity classes.

In following sections we investigate the difficulty that prevents us from completely achieving the two goals simultaneously. In the end, we propose a solution that can partially fulfill Goal 1 and serves our purposes for later development in defining type-2 complexity classes.

2.3 A First Attempt

With the topology $\mathbb{T} \times \mathbb{N}$ and the natural relation between "compactness" and "finite computation", we are now in a position to give type-2 "almost-everywhere" relations a formal definition.

Notation: Let F and G be two functionals of type $\mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$. We use $X_{[F>G]}$ to denote the subset of $\mathcal{T} \times \mathbf{N}$ containing exactly those (f, x) for which $F(f, x) > G(f, x)$. The same usage applies to $X_{[F=X]}, X_{[F \leq G]}$, and so on.

Definition 22 (First Attempt) *Let $F, G : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$ be continuous. We define $F \leq_2^+ G$ by the following:*

$$F \leq_2^+ G \ \text{ if and only if } \ X_{[F>G]} \text{ is compact in } \mathbb{T} \times \mathbb{N}.$$

■

The next theorem states that the type-2 relation $\leq_2^+$ fulfills Goal 2.

Theorem 23 *The type-2 almost-everywhere relation $\leq_{\mathbf{2}}^{+}$ is transitive.*

Proof: Let F, G, and H be continuous type-2 functionals of type $\mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$. Suppose $F \leq_{\mathbf{2}}^{+} G$ and $G \leq_{\mathbf{2}}^{+} H$. Thus, by definition, $X_{[F>G]}$ and $X_{[G>H]}$ are compact in $\mathbb{T} \times \mathbb{N}$. By Theorem 20, $X_{[F>G]}$ and $X_{[G>H]}$ are bounded, and hence so is $X_{[F>G]} \cup X_{[G>H]}$. It is clear that

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

and hence that $X_{[F>H]}$ is bounded. By Theorem 20, $X_{[F>H]}$ is compact. Therefore, $F \leq_{\mathbf{2}}^{+} H$. □

Theorem 24 *Let $F, G : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$ be continuous. If $F \leq_{\mathbf{2}}^{+} G$, 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: Since $F \leq_{\mathbf{2}}^{+} G$, by definition the set $X_{[F>G]}$ is compact in $\mathbb{T} \times \mathbb{N}$. Define $H : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$ by

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

Since F and G are continuous, so is H. Since $X_{[F>G]}$ is compact, so is $H(X_{[F>G]})$ in $\mathbb{N}$. Therefore, H is bounded by a $c \in \mathbf{N}$. □

The notation $\overset{\infty}{\forall}$ is very common in the literature, which quantifies a type-0 variable and is read as "for all but finitely many". For type-1 variables, we abuse the notation $\overset{\boldsymbol{\infty}}{\forall}_{\mathbf{2}}$ by taking it as "for all but compactly many". With Theorem 20, we compare $f \leq^{*} g$ and $F \leq_{\mathbf{2}}^{+} G$ in Table 1. The similarity between $f \leq^{*} g$ and $F \leq_{\mathbf{2}}^{+} G$ suggests that Definition 22 might be a right choice for our purpose. We shall see that this is not the case in the next section.

Assume that F and G are continuous, f, f_0 range over $\mathcal{T}$, and x, x_0 over $\mathbf{N}$.

Type-1: $f, g \in \mathcal{T}$,

$$\begin{aligned} f \le^* g &\equiv \overset{\infty}{\forall} x[f(x) \le g(x)] \\ &\equiv \exists x_0 \forall x[x > x_0 \Rightarrow f(x) \le g(x)]. \end{aligned}$$

Type-2: $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$,

$$\begin{aligned} F \le_{\mathbf{2}}^{+} G &\equiv \overset{\infty}{\forall}_{\mathbf{2}} (f, x)[F(f, x) \le G(f, x)] \\ &\equiv \exists f_0, x_0 \forall f, x[(f > f_0) \vee (x > x_0) \Rightarrow F(f, x) \le G(f, x)]. \end{aligned}$$

Table 1: Comparison Between Type-1 $\le^*$ and Type-2 $\le_{\mathbf{2}}^{+}$

2.3.1 Problems with $\mathbb{T} \times \mathbb{N}$

The nice analogy between the type-1 and type-2 almost-everywhere relations in Table 1 encourages us to believe that Definition 22 may have given a right notion of type-2 almost-everywhere relations. Unfortunately, this is not the case when we consider Goal 1 of Section 2.2.1. The following theorem shows that $\le_{\mathbf{2}}^{+}$ is same as the exactly less relation $\le$. In other words, the notion of $\le_{\mathbf{2}}^{+}$ does not grant any freedom for us in defining type-2 complexity classes and proving theorems on them.

Theorem 25 *Let $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be continuous. If $F \le_{\mathbf{2}}^{+} G$, then $X_{[F>G]}$ is empty.*

Proof: Suppose that $X_{[F>G]}$ is not empty. Let $(f, a) \in X_{[F>G]}$. Since both F and G are continuous, there exists $\sigma \in \mathcal{F}$ such that $\sigma \subset f$ and (σ, a) is a locking fragment of both F and G. Thus, for every $(g, a) \in (\!(\sigma, a)\!)$, $F(g, a) > G(g, a)$, and hence $(\!(\sigma, a)\!) \subseteq X_{[F>G]}$. Since every nonempty open set in $\mathbb{T} \times \mathbb{N}$ is not compact, it follows that $X_{[F>G]}$ is not compact. $\square$

Moreover, Definition 22 does not provide an equivalent concept to that stated in (10). Consider the following example.

Example 26 *Let $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be defined by:*

$$F(f,x) = \begin{cases} 1 & \textit{if } x = 0, \\ 0 & \textit{otherwise.} \end{cases}$$

$$G(f,x) = \begin{cases} 0 & \textit{if } x = 0, \\ 1 & \textit{otherwise.} \end{cases}$$

Both F and G are computable and hence continuous, and both F and G ignore their type-1 inputs. Clearly, $X_{[F>G]} = ((\emptyset, 0))$ which is noncompact in $\mathbb{T} \times \mathbb{N}$. It is also clear that we can write two OTMs for F and G, respectively, and on all $(f, x) \in ((\emptyset, 0))$, each OTM performs the same computation. Thus, by Definition 22, $F \not\leq_{\mathbf{2}}^{+} G$, but there exist OTMs that compute F and G with finitely many computations on the exception set, $X_{[F>G]}$. In fact, one can prove that any $\mathbb{T} \times \mathbb{N}$ open set is noncompact in $\mathbb{T} \times \mathbb{N}$, whereas an $\mathbb{T} \times \mathbb{N}$ open set may be sensible by some finite computations. Thus, Definition 22 is too restricted to capture the notion stated in (10).

2.4 A Second Attempt

Theorem 25 indicates that Definition 22 fails to meet Goal 1 since $\leq$ and $\leq_{\mathbf{2}}^{+}$ are exactly the same relation over continuous type-2 functionals. Also, Example 26 shows that Definition 22 fails to be the same notion stated in (10) on page 49. This forces us to consider coarser topologies that are relative to the concerned functionals. The induced topology should be coarse enough so that a compact set is not necessarily trivial and should be fine enough so that the formalization of the almost-everywhere relation followed in (11) is not stronger than the intuitive statement in (10). In other words, we want to satisfy Goal 1 said on page 49. We give two alternatives in this section and the next. Unfortunately, as we will see, Goals 1 and 2 stated in Section 2.2.1 are difficult to satisfy at the same time.

2.4.1 The Induced Topology $\mathbb{T}_{max}(A)$ on $\mathcal{T} \times \mathbf{N}$

We want to have a coarser topology in such a way the compact sets are not necessarily trivial when continuous functionals are considered. We choose an induced topology that is relative to the concerned continuous functionals in the following definition.

Definition 27 *Let A be a finite set of continuous functionals of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$. Let $\mathbb{T}_{max}(A)$ be the topology on space $\mathcal{T} \times \mathbf{N}$ defined as follows: Take $(\!(\sigma, a)\!)$ as a basic open set if (σ, a) is a minimal fragment that locks every functional in A.* ■

Let $A = \{F_1, F_2, \dots, F_n\}$ be a finite set of continuous functionals of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$. Fix $(f, a) \in \mathcal{T} \times \mathbf{N}$. Suppose that $\sigma \in \mathcal{F}$ with $\sigma \subset f$ and $(\!(\sigma, a)\!)$ is a basic open set of $\mathbb{T}_{max}(A)$. Moreover, suppose that

$$(\sigma_1, a), (\sigma_2, a), \dots, (\sigma_n, a), \tag{12}$$

are minimal locking fragments of $F_1, F_2, \dots, F_n$ on (f, a), respectively. We observe that $\sigma = \bigcup_{1 \le i \le n} \sigma_i$ and (σ, a) is a locking fragment for every functional in A but may not be the minimal locking fragment for some functional in A. This explains the use of the subscript *max* in the notation of the topology.

One can see that $\mathbb{T}_{max}(A)$ is the smallest topology on $\mathcal{T} \times \mathbf{N}$ such that each $F \in A$ is continuous with respect to this topology.

We henceforth write $\mathbb{T}_{max}(\{F_1, F_2, \dots\})$ as $\mathbb{T}_{max}(F_1, F_2, \dots)$ for simplicity.

Definition 28 *Let $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be continuous. We define $F \leq_{\mathbf{2}}^{\oplus} G$ by:*

$$F \leq_{\mathbf{2}}^{\oplus} G \ \text{ if and only if } \ X_{[F>G]} \text{ is compact in } \mathbb{T}_{max}(F, G).$$

■

Let us consider the two functionals in Example 26 again. According to the definition, the following collection forms a basis for $\mathbb{T}_{max}(F, G)$:

$$\left\{ (\!(\emptyset, 0)\!), (\!(\emptyset, 1)\!), (\!(\emptyset, 2)\!), \ldots \right\}.$$

Thus, any open cover for $X_{[F>G]}$ must contain $(\!(\emptyset, 0)\!)$, and hence a finite sub-cover for $X_{[F>G]}$ exists. Therefore, $X_{[F>G]}$ is compact in $\mathbb{T}_{max}(F, G)$. By definition, we have $F \leq_{\mathbf{2}}^{\oplus} G$.

2.4.2 Problems with $\mathbb{T}_{max}(F, G)$

The following theorem shows that the compact sets in these much coarser topologies are too big in the sense that $\leq_{\mathbf{2}}^{\oplus}$ fails to be a transitive relation.

Theorem 29 *There are continuous F, G and $H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, such that $F \leq_{\mathbf{2}}^{\oplus} G$ and $G \leq_{\mathbf{2}}^{\oplus} H$ but $F \not\leq_{\mathbf{2}}^{\oplus} H$.*

Proof: Consider the following three type-2 functionals:

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

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

$$H(f,x) = \begin{cases} \left\lceil \frac{f(0)-1}{f(0)+1} \right\rceil & \text{if } x = 0; \\ 2 + f(0) & \text{otherwise.} \end{cases}$$

It is clear that F, G and H are continuous, and $G \leq_{\mathbf{2}}^{\oplus} H$. To see that $F \leq_{\mathbf{2}}^{\oplus} G$, we observe that F and G ignore the type-1 input and hence the basic open sets of $\mathbb{T}_{max}(F,G)$ are $(\!(\emptyset, 0)\!), (\!(\emptyset, 1)\!), (\!(\emptyset, 2)\!), \ldots$. We know that $X_{[F>G]} = (\!(\emptyset, 0)\!)$, which is compact in $\mathbb{T}_{max}(F,G)$. To see this, consider any open cover $\mathcal{O}$ of $X_{[F>G]}$. $\mathcal{O}$ must contain at least one open set that is a superset of $(\!(\emptyset, 0)\!)$. Select this set as the singleton subcover of $(\!(\emptyset, 0)\!)$. Thus, $F \leq_{\mathbf{2}}^{\oplus} G$.

If we compare F and H, we find that the exception set $X_{[F>H]}$ is also $(\!(\emptyset, 0)\!)$. However, $(\!(\emptyset, 0)\!)$ is not compact in $\mathbb{T}_{max}(F,H)$. To see this, we observe that H converges at $f(0)$ for every input, and thus, the family of the basic open sets of $\mathbb{T}_{max}(F,H)$ contains

$$(\!(\{(0,0)\}, 0)\!), (\!(\{(0,1)\}, 0)\!), (\!(\{(0,2)\}, 0)\!), \ldots\ldots \tag{13}$$

The open sets above form an open cover for $X_{[F>H]}$, which has no finite subcover that covers $X_{[F>H]}$. Therefore, $F \not\leq_{\mathbf{2}}^{\oplus} H$. $\square$

Note that the example in the proof above does not work in the topology $\mathbb{T} \times \mathbb{N}$. This is because the exception set $X_{[F>G]}$ is not compact in $\mathbb{T} \times \mathbb{N}$, since the sets in (13) are also open in $\mathbb{T} \times \mathbb{N}$, and hence (13) forms an open cover for $X_{[F>G]}$ without a finite sub-cover.

2.5 A Third Attempt and Limited Success

By using the definition of $\leq_{\mathbf{2}}^{\oplus}$, we face the problem that a bigger functional may determine a finer topology, which results in a loss of transitivity. In this section we introduce another alternative to define a coarser induced topology to bring back the transitivity property. However, having transitivity is just a part of Goal 2 stated in Section 2.2.1. The primary purpose of having transitivity is to preserve the basic non-decreasing hierarchy among type-2 complexity classes. However, both $\mathbb{T}_{max}$ and the next alternative do not solve this problem directly. Nevertheless, our next class of induced topologies gives us some room to pin down the problem when we actually define type-2 complexity classes in the following chapters. We shall come back to this issue in the next chapter.

2.5.1 The Induced Topology $\mathbb{T}_{min}(A)$ on $\mathcal{T} \times \mathbf{N}$

Definition 30 *Let $A = \{F_1, F_2, \ldots, F_n\}$ be a finite set of continuous functionals of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$. Let $\mathbb{T}_{min}(A)$ be the topology on space $\mathcal{T} \times \mathbf{N}$ defined as follows: for each $(f, a) \in \mathcal{T} \times \mathbf{N}$, let $(\!(\sigma, a)\!)$ be a basic open set of $\mathbb{T}_{min}(A)$ if (σ, a) is of the following form*

$$(\!(\sigma, a)\!) = (\!(\sigma_1, a)\!) \cap (\!(\sigma_2, a)\!) \cdots \cap (\!(\sigma_n, a)\!),$$

where each (σ_i, a) is the minimal locking fragment of F_i on (f, a). ■

We express $\mathbb{T}_{min}(\{F_1, F_2, \ldots\})$ as $\mathbb{T}_{min}(F_1, F_2, \ldots)$ for simplicity.

Let $A = \{F_1, F_2, \ldots, F_n\}$ be a finite set of continuous functionals of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$. Suppose that $(\sigma_1, a), (\sigma_2, a), \ldots, (\sigma_n, a)$ are minimal locking fragments of $F_1, F_2, \ldots, F_n$ on $(f, a) \in \mathcal{T} \times \mathbf{N}$, respectively. We observe that (σ, a), where

$\sigma = \bigcap_{1 \leq i \leq n} \sigma_i$, may not be a locking fragment for any functional in A. Consider the following example.

Example 31 *Define $F_0, F_1 : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ by*

$$F_0(f, x) = \begin{cases} f(0) & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{cases}$$

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

Consider the set $(\!(\emptyset, 0)\!)$. We want to show that $(\!(\emptyset, 0)\!)$ is open in each of $\mathbb{T}_{min}(F_0, F_1)$, $\mathbb{T}_{min}(F_0)$, and $\mathbb{T}_{min}(F_1)$, respectively. We also want to show that $(\!(\emptyset, 0)\!)$ is compact in $\mathbb{T}_{min}(F_0, F_1)$ but not so in $\mathbb{T}_{min}(F_0)$, nor in $\mathbb{T}_{min}(F_1)$.

Fix any $f \in \mathcal{T}$. Let $\sigma_0 = \{(0, f(0))\}$ and $\sigma_1 = \{(1, f(1))\}$. Since $(\sigma_0, 0)$ and $(\sigma_1, 0)$ are minimal locking fragments of F_0 and F_1, respectively, and $\sigma_0 \cap \sigma_1 = \emptyset$, it follows that $(\!(\emptyset, 0)\!)$ is a basic open set of the topology $\mathbb{T}_{min}(F_0, F_1)$. Thus, the following collection of open sets forms a basis for $\mathbb{T}_{min}(F_0, F_1)$:

$$\left\{ (\!(\emptyset, 0)\!), (\!(\emptyset, 1)\!), (\!(\emptyset, 2)\!), \ldots \right\}.$$

Clearly, $(\!(\emptyset, 0)\!)$ is compact in $\mathbb{T}_{min}(F_0, F_1)$. For $\mathbb{T}_{min}(F_0)$, consider the fact that

$$(\!(\emptyset, 0)\!) = \bigcup_{i \in \mathbf{N}} (\!(\sigma_0^i, 0)\!), \quad \text{where } \sigma_0^i = \{(0, i)\}.$$

Clearly, for each $i \in \mathbf{N}$, $(\!(\sigma_0^i, 0)\!)$ is a basic open set of $\mathbb{T}_{min}(F_0)$, and hence $(\!(\emptyset, 0)\!)$ is open in $\mathbb{T}_{min}(F_0)$. However, $(\!(\emptyset, 0)\!)$ is not compact in $\mathbb{T}_{min}(F_0)$ due to the fact that

$$\left\{ (\!(\sigma_0^0, 0)\!), (\!(\sigma_0^1, 0)\!), (\!(\sigma_0^2, 0)\!), \ldots \right\}$$

is an open cover without any finite sub-cover for $(\!(\emptyset, 0)\!)$. For $\mathbb{T}_{min}(F_1)$, we can apply the same argument to conclude that $(\!(\emptyset, 0)\!)$ is open but not compact in $\mathbb{T}_{min}(F_1)$.

The following three lemmas establish some basic properties of the $\mathbb{T}_{min}(\cdot)$ topologies.

Lemma 32 *Let A and B be two finite sets of continuous functionals of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$. If $A \subseteq B$, then $\mathbb{T}_{min}(B) \subseteq \mathbb{T}_{min}(A)$.*

Proof: Let A and B be two finite sets of continuous functionals such that $A \subseteq B$. Fix $(f, a) \in \mathcal{T} \times \mathbf{N}$. Consider the fact that, for every $n \in \mathbf{N}$,

$$(\sigma_0 \cap \sigma_1 \cdots \cap \sigma_n \cap \sigma_{n+1}) \;\subseteq\; (\sigma_0 \cap \sigma_1 \cdots \cap \sigma_n), \quad \text{where } \sigma_i \in \mathcal{F}, 0 \leq i \leq n+1.$$

Thus, if $\sigma_A \subset f$, $\sigma_B \subset f$, and $(\!(\sigma_A, a)\!)$ and $(\!(\sigma_B, a)\!)$ are two basic open sets of $\mathbb{T}_{min}(A)$ and $\mathbb{T}_{min}(B)$, respectively, then $\sigma_B \subseteq \sigma_A$. Therefore, $(\!(\sigma_A, a)\!) \subseteq (\!(\sigma_B, a)\!)$. In fact, $(\!(\sigma_B, a)\!)$, a basic open set of $\mathbb{T}_{min}(B)$, is either equal to $(\!(\sigma_A, a)\!)$ or an infinite union of some basic open sets of $\mathbb{T}_{min}(A)$. In other words, $\mathbb{T}_{min}(A)$ is finer that $\mathbb{T}_{min}(A)$. Thus, $\mathbb{T}_{min}(B) \subseteq \mathbb{T}_{min}(A)$ □

Lemma 33 *Let A be a finite set of continuous functionals of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$. If $S \subseteq \mathcal{T} \times \mathbf{N}$ is open in $\mathbb{T}_{min}(A)$, then S is closed in $\mathbb{T}_{min}(A)$.*

Proof: Fix any open set $S = \bigcup_{i \geq 0} (\!(\sigma_i, a_i)\!)$, where for every $i \geq 0$, $\sigma_i \in \mathcal{F}$, $a_i \in \mathbf{N}$, and $(\!(\sigma_i, a_i)\!)$ is a basic open set in $\mathbb{T}_{min}(A)$. Suppose that $(f, a) \in \mathcal{T} \times \mathbf{N}$ and $(f, a) \notin S$. Let $\tau \subset f$ and $(\!(\tau, a)\!)$ be a basic open set in $\mathbb{T}_{min}(A)$. Such τ exists because A is a set of total continuous type-2 functionals. Thus, it must be the case that either, for every $i \geq 0$, $a \neq a_i$ or there exists $x \in \mathbf{N}$ such that, $x \in \mathsf{dom}(\sigma_i)$ and $\sigma_i(x) \neq \tau(x)$, because otherwise $(f, a) \in S$. Since $(f, a) \in (\!(\tau, a)\!)$ and $(\!(\tau, a)\!) \cap S = \emptyset$, (f, a) is not an accumulation point of S. By the contrapositive, all accumulation points of S are in S. Therefore, S is closed. □

Lemma 34 *Let A be a finite set of continuous functionals of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$. If $X \subseteq \mathcal{T} \times \mathbf{N}$ is compact in $\mathbb{T}_{min}(A)$, then every $S \subseteq X$ is also compact in $\mathbb{T}_{min}(A)$.*

Proof: Consider the topology $\mathbb{T}_{min}(A)$ and let X be compact in $\mathbb{T}_{min}(A)$, and $S \subseteq X$. Given $\mathcal{O}$ any open cover of S, let U be the union of the open sets in $\mathcal{O}$. Thus, U is open. By Lemma 33, U is closed, and hence U^c, the complement of U, is open. Thus, $\{U^c\} \cup \mathcal{O}$ is an open cover for X. Since X is compact, we can extract a finite subcover, $\mathcal{O}'$, from $\{U^c\} \cup \mathcal{O}$ that covers X. It is clear that $\mathcal{O}'$ also covers S. If U^c is in $\mathcal{O}'$, then we can remove it from $\mathcal{O}'$ and the remaining subcover still covers S as well because $U^c \cap S = \emptyset$. Therefore, there is a finite subcover of $\mathcal{O}$ for S, and hence S is compact. □

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

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

Definition 35 *Let $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be continuous. We define $F \leq_2^* G$ by:*

$$F \leq_2^* G \;\; \textit{if and only if} \;\; X_{[F>G]} \textit{ is compact in } \mathbb{T}_{min}(F,G).$$

■

Recall Goals 1 and 2 from Section 2.2.1 on page 49. We shall show that the above definition brings back the transitivity which is a part of Goal 2, but it only partially satisfies Goal 1. Recall the two functionals, F_0 and F_1, given in Example 31 on page 58 and the basis of $\mathbb{T}_{min}(F_0, F_1)$:

$$\left\{ ((\emptyset, 0)), ((\emptyset, 1)), ((\emptyset, 2)), \ldots \right\}.$$

According to the definition, we have

$$X_{[F_0>F_1]} = \left\{ (f, 0) \;\middle|\; f \in \mathcal{T} \text{ with } f(0) > f(1) \right\}.$$

It is clear that $X_{[F_0>F_1]}$ is compact in $\mathbb{T}_{min}(F_0, F_1)$, because every $(f, a) \in X_{[F_0>F_1]}$ must be an element of the basic open set $(\!(\emptyset, 0)\!)$, and no other basic open set can contain such (f, a). Thus, any open cover for $X_{[F_0>F_1]}$ must contain $(\!(\emptyset, 0)\!)$. Thus, $F_0 \leq_{\mathbf{2}}^* F_1$. However, any two OTMs that compute F_0 and F_1, respectively, perform infinitely many computations on $X_{[F_0>F_1]}$. For example, if $f, g \in \mathcal{T}$ with $f(0) \neq g(0)$, then the two computations of any OTM that computes F_0 on $(f, 0)$ and $(g, 0)$, respectively, are different. In next chapter we solve this problem by removing one of the two functionals from determining the induced topology. Here we modify our Goal 1 as a compromise but it is suitable for our purposes in defining type-2 complexity classes.

Compromised Goal 1: Suppose that type-2 functionals F and G are computable. We want $F \leq_{\mathbf{2}}^* G$ to be the case that there is an OTM $\widehat{M}_e$ such that, there are only finitely many computation of $\widehat{M}_e$ on the exception set $X_{F>G}$.

If such an OTM $\widehat{M}_e$ in Compromised Goal 1 exists, then we can use $\widehat{M}_e$ to realizes $F > G$ on the exception set by only finitely many computations. In the followings chapter, we will see how such an OTM that realizes $F > G$ by finitely many computations provides a way to patch a $\widehat{\varphi}$-program with cost function F that is not bounded by cost bound G on an exception set.

Theorem 36 *The type-2 almost-everywhere relation $\leq_{\mathbf{2}}^*$ is transitive.*

Proof: Let $F, G, H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be continuous. Suppose $F \leq_{\mathbf{2}}^* G \leq_{\mathbf{2}}^* H$. Thus, $X_{[F>G]}$ and $X_{[G>H]}$ are compact in $\mathbb{T}_{min}(F, G)$ and $\mathbb{T}_{min}(G, H)$, respectively. Clearly, $X_{[F>G]} \cup X_{[G>H]}$ is compact in $\mathbb{T}_{min}(F, G, H)$. Since $X_{[F>H]} \subseteq X_{[F>G]} \cup X_{[G>H]}$, by Lemma 34, $X_{[F>H]}$ is compact in $\mathbb{T}_{min}(F, G, H)$. By Lemma 32, $X_{[F>H]}$ is compact in $\mathbb{T}_{min}(F, H)$. Therefore, $F \leq_{\mathbf{2}}^* H$. $\square$

We will deal with the basic hierarchy requirement in Goal 2 when we actually define a type-2 time bound in the next Chapter.

Notations: $\overset{\infty}{\forall}_{\mathbf{2}}$ **and** $\overset{\infty}{\exists}_{\mathbf{2}}$ We choose $\leq_{\mathbf{2}}^{*}$ to be our standard notion of almost-everywhere less or equal relation over type-2 functionals. For simplicity, we simply use $\mathbb{T}(\cdot)$ to denote $\mathbb{T}_{min}(\cdot)$. Moreover, two type-2 analogs of $\overset{\infty}{\forall}$ and $\overset{\infty}{\exists}$ are defined by

Definition 37 *1. For continuous functionals $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, we say that*

$$\overset{\infty}{\forall}_{\mathbf{2}}\,(f,x)[F(f,x) \leq G(f,x)] \quad \textit{if and only if}$$
$$\left\{(f,x) \,\middle|\, F(f,x) \leq G(f,x)\right\} \ \textit{is compact in } \mathbb{T}(F,G).$$

2. For continuous functionals $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, we say that

$$\overset{\infty}{\exists}_{\mathbf{2}}\,(f,x)[F(f,x) \leq G(f,x)] \quad \textit{if and only if}$$
$$\left\{(f,x) \,\middle|\, F(f,x) \leq G(f,x)\right\} \ \textit{is noncompact in } \mathbb{T}(F,G).$$

■

Thus, for continuous functionals $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, we have

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

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

Chapter 3

Type-2 Time Bounds

A dynamic complexity measure is a way of gauging the cost of a computation in terms of certain resources. There are many specific complexity measures studied, e.g., time, space, number of coin-flips, flowchart measure, read-write head reversals, and so on. For type-1 computations, Blum's abstract complexity axioms [Blu67] successfully isolate key features of many of the specific measures.

For our study of type-2 computational complexity we focus simply on time complexity since the time cost is probably the simplest and most intuitive dynamic resource. In this chapter we define a class of functions that are suitable to serve as time bounds for computations of OTMs.

Notations and Conventions

Recall that $\mathcal{T}$ denotes the set of total functions of type $\mathbf{N} \to \mathbf{N}$ and $\mathcal{F}$ the set of finite functions. Let f and g range over $\mathcal{T}$; σ, τ, and ι range over $\mathcal{F}$; and e, x, y, a, and b range over $\mathbf{N}$. Thus, we do not explicitly specify their universes in quantified formulas for simplicity. Also, recall that $\widehat{M}_e$ denotes the OTM with index e, which computes the functional $\widehat{\varphi}_e : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$. The time cost function of $\widehat{M_e}$ is denoted by $\widehat{\Phi}_e$. Also, recall from Section 1.4 that the time cost is simply the number of steps performed by the OTM, and we have two cost models from which to choose. For Turing machines and type-1 computable functions, we use M_e, φ_e, and Φ_e in the same manner. Moreover, we introduce the following notation for convenience.

- For $\sigma \in \mathcal{F}$, let $\mathsf{dom}(\sigma)$ denote the domain of σ, and $\mathsf{img}(\sigma)$ the image of σ.

- If $\sigma \neq \emptyset$, we define $\|\sigma\|$ by

$$\|\sigma\| = \sum_{i \in \mathsf{dom}(\sigma)} (|i| + |\sigma(i)| + 2).$$

  If $\sigma = \emptyset$, by convention, we let $\|\sigma\| = 2$.

- Let $\sigma^{\sim 0}$ denote the total zero extension of σ:

$$\sigma^{\sim 0}(x) = \begin{cases} \sigma(x) & \text{if } x \in \mathsf{dom}(\sigma); \\ 0 & \text{otherwise.} \end{cases}$$

- For each $f \in \mathcal{T}$ and $n \in \mathbf{N}$, let $f_{[n]}$ denote the initial n-element fragment of f:

$$f_{[n]} = \left\{ \langle x, f(x) \rangle \;\middle|\; x < n \right\}.$$

  Thus, $f_{[n]} \in \mathcal{F}$. Similarly, for each $\sigma \in \mathcal{F}$ and $n \in \mathbf{N}$, define $\sigma_{[n]}$ by

$$\sigma_{[n]} = \left\{ \langle x, \sigma(x) \rangle \;\middle|\; x < n \right\}.$$

  Note that $\sigma_{[n]}$ is not necessarily an initial fragment of some total function.

3.1 Complexity Classes and Time Bounds

We follow Hartmanis and Stearns's conventions in [HS65] in defining type-1 complexity classes. That is, for each $t \in \mathcal{R}$, the complexity class $\mathcal{C}(t)$ is defined by:

$$\mathcal{C}(t) = \left\{ f \in \mathcal{R} \,\middle|\, \exists e \left[\varphi_e = f \text{ and } \overset{\infty}{\forall} x \left(\Phi_e(x) \leq t(|x|) \right) \right] \right\}. \tag{14}$$

We say that t is the name of the complexity class $\mathcal{C}(t)$. For $f \in \mathcal{C}(t)$, we say that the computational complexity of f is bounded by the bound t. For example, if we use deterministic TMs as the computation device and take the time cost as the complexity measure, then the complexity class determined by t is denoted by $\mathit{DTIME}(t)$. Thus, $f \in \mathit{DTIME}(t)$ means that f is recursive and that there is a TM M_e computing f provided M_e on all $x \in \mathbf{N}$ but finitely many halts within $t(|x|)$ steps. Likewise, $\mathit{DSPACE}(t)$ is determined in the same manner except that the complexity measure is changed to the amount of tape space used. Similarly, $\mathit{NTIME}(t)$ and $\mathit{NSPACE}(t)$ use time and space as complexity measures, respectively, but we change the computing device to nondeterministic Turing machine model.

Except for totality and computability, there is almost no further restriction on resource-bound functions in defining type-1 complexity classes.[1] We will see later how we lose this freedom in defining a proper type-2 time bound.

For type-2 complexity classes, Seth suggested two proposals in [Set94]. In fact, the idea behind the two proposals can be found in some work on machine characterizations for type-2 complexity classes. We state the two proposals in the following two definitions.

Definition A: Let $t : \mathbf{N} \to \mathbf{N}$ be recursive. Let $\mathit{DTIME}_A(t)$ denote the set of type-2 functionals such that, for every functional $F \in \mathit{DTIME}_A(t)$, F is total and there

[1] Constructability (honesty) is another restriction we impose on resource-bound functions. See Section 5.3 for the reason we want this restriction.

exists an OTM $\widehat{M}_e$ that computes F and, on *every* $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{M}_e$ halts within $t(m)$ steps, where $m = |\mathsf{max}(\{x, Q\})|$ and Q is the set of all answers returned from the oracle during the course of the computation. ■

Definition B: Let $H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be computable. Let $DTIME_B(H)$ denote the set of type-2 functionals such that, for every functional $F \in DTIME_B(H)$, F is total and there exists an OTM $\widehat{M}_e$ that computes F and, on *every* $(f, x) \in (\mathbf{N} \to \mathbf{N}) \times \mathbf{N}$, $\widehat{M}_e$ halts within $H(f, x)$ steps. ■

The two definitions above for type-2 complexity classes are lifted from the definition of type-1 complexity classes. However, the two naive analogs introduce some problems that do not exist in type-1 computations.

Discussion for Definition A: The main difficulty of Definition A is that the set Q is unknown when the computation starts. Although Q is determined by a computation, it is not computable in general, as can be seen by a simple diagonalization argument. Thus, we cannot have an effective procedure to obtain Q in advance and compute the time bound. As a consequence of this difficulty, it is hard to build a recursive presentation of such a complexity class. The only possible way out is to recalculate the bound dynamically upon each oracle query made during the course of the computation. For example, a POTM is an OTM bounded by a polynomial in this manner (see Section 1.3). However, this still does not solve our problem, since a POTM does not need to be total, as shown in the following theorem.

Theorem 38 *There is a POTM that is not total.*

Proof: Consider the $\widehat{\varphi}$ program in Figure 2 for F. Clearly, if the input function $f \in \mathcal{T}$ is strictly increasing in size, then for every $x \in \mathbf{N}$, $F(f, x)$ diverges. However, we can

```
Program F
      input f : T, x : N;
      i ⟵ 0;
      while |f(i)| < |f(i + 1)| do
            i ⟵ i + 1;
      end while;
      output f(i);
End program
```

Figure 2: A Partial POTM Computable Functional

construct an OTM $\widehat{M_e}$ that computes F and is bounded by a polynomial $p \in O(n^2)$. To see this, let $\widehat{M_e}$ be the OTM that computes F, and, on every $(f, x) \in \mathcal{T} \times \mathbf{N}$, let $\widehat{M_e}$ check the values $f(0), f(1), \ldots$ in order. Let $\sigma_t \in \mathcal{F}$ denote the collection of oracle queries from the beginning of the computation up to cost t. Fix cost $t \in \mathbf{N}$. Suppose $\mathsf{max}(\mathsf{dom}(\sigma_t)) = i$. According to our conventions, the number of steps $\widehat{M_e}$ has performed up to cost t is simply t. Thus, $t = k \cdot \|\sigma_t\|$ for some constant k. Without loss of generality, assume $x = 0$. By assumptions, $|\mathsf{max}(x, \mathsf{img}(\sigma_t))| = |f(i)|$. Let $\ell = |f(i)|$. Since f is strictly increasing in size, $\ell \geq i$. Thus,

$$k \cdot \|\sigma_t\| = k \cdot \Big((|0| + |f(0)|) + (|1| + |f(1)|) + \cdots + (|i| + |f(i)|) + (i + 1) \Big) \leq k\ell^2.$$

Therefore, $\widehat{M_e}$ is bounded by a polynomial in $O(n^2)$. □

Clearly, the problem noted in Theorem 38 is a nuisance, since a complexity class under Definition A is not necessarily recursive representable even if the resource-bound function t is just a small polynomial. This suggests that we should impose more restrictions on resource-bound functions for type-2 complexity classes.

Discussion for Definition B: We further modify Definition B along the line of defining type-1 complexity classes, where the "*size*" of the input is taken for the resource-bound functions. Let $f \in \mathcal{T}$. Recall Kapron and Cook's [KC91, KC96] length function $|f| : \mathbf{N} \to \mathbf{N}$,

$$|f| = \lambda a.\mathsf{max}(\left\{ |f(x)| \;\middle|\; |x| \leq a \right\}).$$

Also, recall the definition of $\overset{\infty}{\forall}_{\mathbf{2}}$ described in Section 2.5, where the type-2 almost-everywhere relation $\leq_{\mathbf{2}}^{*}$ is involved. By using the length functions together with the quantifier $\overset{\infty}{\forall}_{\mathbf{2}}$, Definition B can be modified into a closer analog to the definition of type-1 complexity classes as follows.

Definition C: Let $H : (\mathbf{N} \to \mathbf{N}) \times \mathbf{N} \to \mathbf{N}$ be computable. Define

$$\begin{aligned} &\mathit{DTIME}_C(H) = \\ &\quad \left\{ F : \mathcal{T} \times \mathbf{N} \to \mathbf{N} \;\middle|\; \exists e \left[\widehat{M}_e = F \text{ and } \overset{\infty}{\forall}_{\mathbf{2}} \, (f, x) \left(\widehat{\Phi}_e(f, x) \leq H(|f|, |x|) \right) \right] \right\}. \end{aligned}$$

■

Recall Kapron and Cook's second-order polynomials from Section 1.3. Clearly, a second-order polynomial is a computable functional of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$. As the fact that restricting t in (14) on page 65 to a polynomial yields a subclass of *PTIME*,[2] Kapron and Cook [KC91, KC96] prove that restricting H of Definition C to one of their second-order polynomials yields a subclass of BFF_2. This not only provides evidence to consider BFF_2 as a likely feasible class, but also suggests that Definition C may be a likely way to define type-2 complexity classes. Seth [Set94] suggests that Definition C may result in the type-2 analog of the classical complexity theory.

However, we do not think using type-2 functionals as time bounds is proper for two reasons. First, because every terminating computation is finite, only a finite part of the type-1 input contributes to the computation. Thus, it seems unnecessary to let the time bound "know" the entire type-1 input. In other words, the time bound

[2] By the Union theorem [MM69], *PTIME* itself is a complexity class under the definition in (14).

should not be based on information that is irrelevant to the computation. Second, the length function is difficult to compute. That is, computing $(f, x) \mapsto |f|(|x|)$ is not basic feasible. Thus, a clocked OTM will become inefficient when the cost of the computation of the clock is considered.

3.2 Type-2 Time Bounds ($\mathbf{T_2TB}$)

In this section we define a class of functions that, as we believe, are suitable to serve as time bounds for type-2 computation. We call them Type-2 Time Bounds and we will use them to clock our OTMs.

Intuitively, type-2 time bounds must satisfy the following requirements.

1. Every type-2 time bound must be computable.
2. Every OTM clocked by any type-2 time bound must be total.
3. Every total OTM must be computed by some clocked OTM.
4. The clock must update the time bound dynamically during the course of the computation. In other words, the bound must depend on information gathered over the course of the computation that is being clocked. ■

Similar to Definition 15 on page 42, we define *locking fragments* and *minimal locking fragments* of $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ as follows.

Definition 39 *Let $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ and $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$.*

1. *We say that (σ, a) is a **locking fragment** of β if and only if*

$$(\forall \tau \supseteq \sigma) \left[\beta(\sigma, a) = \beta(\tau, a) \right].$$

2. *We say that (σ, a) is a* ***minimal locking fragment*** *of β if and only if (σ, a) is a locking fragment of β and, for every $\tau \in \mathcal{F}$ with $\tau \subset \sigma$, (τ, a) is not a locking fragment of β.* ■

If (σ, a) is a locking fragment of β, we say that β is *locked* on (σ, a), and we denote this by $\beta(\sigma, a)\downarrow$. Note that for a locking fragment (σ, a), $\mathsf{dom}(\sigma)$ is not necessarily an initial fragment of $\mathbf{N}$. If (σ, a) is a minimal locking fragment of β and $\sigma \neq \emptyset$, then we have

$$(\forall \tau \subset \sigma)(\exists\, \iota \supseteq \tau)\left[\beta(\iota, a) \neq \beta(\sigma, a)\right].$$

In the following definitions, we state four properties that are expected for a function $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ to serve as a clock for OTMs.

Definition 40 *Let $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$.*

1. **Nontriviality:** *We say that β is* ***nontrivial*** *if for every $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$,*

$$\beta(\sigma, a) \geq |a| + 1.$$

2. **$\mathcal{F}$-monotonicity:** *We say that β is* ***$\mathcal{F}$-monotone*** *if and only if, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$ and $\sigma \in \mathcal{F}$ such that $\sigma \subset f$, we have*

$$\beta(\sigma, x) \leq \lim_{\tau \to f} \beta(\tau, x).$$

3. **Strong $\mathcal{F}$-monotonicity:** *We say that β is* ***strongly $\mathcal{F}$-monotone*** *if and only if, for every $\sigma, \tau \in \mathcal{F}$ and $a \in \mathbf{N}$,*

$$\sigma \subseteq \tau \Rightarrow \beta(\sigma, a) \leq \beta(\tau, a).$$

4. **Convergence:** *We say that β is* ***convergent*** *if and only if, for every $(f, a) \in \mathcal{T} \times \mathbf{N}$, there exists $\sigma \subset f$ such that, for all $\tau \supseteq \sigma$, $\beta(\sigma, a) = \beta(\tau, a)$. That is,*

$$\forall(f, a)\exists\sigma \subset f \left[\lim_{\tau \to f} \beta(\tau, a) \;=\; \beta(\sigma, a) \right].$$

■

Definition 41 (Type-2 Time Bound) *We say that $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ is a type-2 time bound if and only if β is (i) computable, (ii) nontrivial, (iii) $\mathcal{F}$-monotone, and (iv) convergent.* ■

Definition 42 (Strong Type-2 Time Bound) *We say that $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ is a strong type-2 time bound if and only if β is (i) computable, (ii) nontrivial, (iii) strongly $\mathcal{F}$-monotone, and (iv) convergent.* ■

We use $\mathbf{T_2TB}$ to denote the set of type-2 time bounds, and use α, β, and γ as variables ranging over $\mathbf{T_2TB}$.

Since there is a canonical indexing for finite functions that is a 1-1 correspondence between $\mathcal{F}$ and $\mathbf{N}$, we can view each $\beta \in \mathbf{T_2TB}$ as a type-1 function from $\mathbf{N} \times \mathbf{N}$ to $\mathbf{N}$. On the other hand, each type-2 time bound β determines a computable type-2 functional $F_\beta : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ (see Section 3.3 for details).

Recall from Section 1.4 that we require each OTM to scan its entire type-0 input at the beginning of its computation. Thus, a clock must be nontrivial in order to allow an OTM to scan its type-0 input. In other words, nontriviality prevents empty complexity classes. If a β is strongly $\mathcal{F}$-monotone, then the bound provided by β will not shrink during the course of the computation. However, we fail to see a significant reason for restricting our type-2 time bounds from shrinking the budget during the course of the computation. Nevertheless, $\mathcal{F}$-monotonicity requires the value of β to be bounded by its value in the limit. This requirement prevents the loss of some intuitive understanding of a complexity class determined by β. Clearly, if β is strongly $\mathcal{F}$-monotone, then β is $\mathcal{F}$-monotone. Finally, if β is convergent, then every OTM clocked by β terminates on every input. In such a way, we have a chance to recursively representing the functionals in a type-2 complexity class. The points we just mentioned will become clear after we introduce our clocking schema in Section 3.4

Consider the following examples.

Example 43 *Fix some integer $k > 1$. Define $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ by*

$$\beta(\sigma, x) = \sum_{0 \le i \le x \,\wedge\, i \in \mathsf{dom}(\sigma)} 2 \cdot \sigma(i) + (x+1) \cdot |x| + k.$$

Example 44 *Define $f : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ by*

$$h(\sigma, x) = \begin{cases} \|\sigma\| + |x| + 1 & \textit{if } \mathsf{img}(\sigma) = \{0\}; \\ |\sigma(i)| + |x| + 1 & \textit{if } i \textit{ is the least number such that } \sigma(i) \neq 0. \end{cases}$$

One can easily see that β defined in Example 43 is a type-2 time bound because it is computable, nontrivial, strongly $\mathcal{F}$-monotone, and convergent. On the contrary, the function h defined in Example 44 is neither convergent nor $\mathcal{F}$-monotone.

Theorem 45 $\mathbf{T_2TB}$ *is not recursively representable.*

Proof: We omit the proof, which can be easily obtained by a standard diagonalization. □

3.2.1 Locking Detectable Type-2 Time Bounds

By definition, a type-2 time bound is convergent and thus, for any $(f, a) \in \mathcal{T} \times \mathbf{N}$, a locking fragment must exists. For some type-2 time bounds, we can decide if a given (σ, a) is a locking fragment. We formulate this property in the following two definitions.

Definition 46 (Locking Dectectors) *Let $\beta \in \mathbf{T_2TB}$. Suppose $\ell : \mathcal{F} \times \mathbf{N} \to \{0,1\}$ has the following four properties:*

1. *ℓ is computable.*

2. *ℓ is strongly $\mathcal{F}$-monotone.*

3. $\forall(\sigma, x) \in \mathcal{F} \times \mathbf{N} \left[\ell(\sigma, x) = 1 \implies \beta(\sigma, x) \downarrow \right]$.

4. $\forall(f, x) \in \mathcal{T} \times \mathbf{N} \left[\lim_{\sigma \to f} \ell(\sigma, x) = 1 \right]$.

Then, we say that ℓ is a ***locking detector*** *of β.* ■

Let ℓ be a locking detector of β. Note that, since the inverse of the third property above need not hold for ℓ, therefore, while a minimal locking fragment of ℓ must be a locking fragment of β, it may not be a *minimal* locking fragment of β.

Definition 47 (Locking Detectable $\mathbf{T_2TB}$) *We say that $\beta \in \mathbf{T_2TB}$ is a locking-detectable type-2 time bound if β has a locking detector.* ■

One can see that the type-2 time bound given in Example 43 is locking detectable via the following locking detector:

$$\ell(\sigma, x) = \begin{cases} 1 & \text{if } \{0, \ldots, x\} \subseteq \mathsf{dom}(\sigma); \\ 0 & \text{otherwise.} \end{cases}$$

Many proofs in the following chapters will be significantly simplified by restricting $\mathbf{T_2TB}$ to locking-detectable type-2 time bounds. However, we will see later that not every type-2 time bound is locking detectable by a computable function. On the other hand, Theorem 88 in the next chapter states that we will not lose any type-2 complexity classes by restricting $\mathbf{T_2TB}$ to locking-detectable type-2 time bounds.

3.2.2 Oracle Turing Machines and $\mathbf{T}_2\mathbf{TB}$

For each OTM $\widehat{M}_e$ that halts on every $(f, x) \in \mathcal{T} \times \mathbf{N}$, we can obtain a fairly natural type-2 time bound $\beta_{\widehat{M}_e}$ as follows. On input $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, $\beta_{\widehat{M}_e}$ simulates $\widehat{M}_e$ step by step on input x and uses the information of σ to answer the oracle queries posed during the computation of $\widehat{M}_e$. Finally, $\beta_{\widehat{M}_e}$ outputs the number of steps simulated up to the point where one of the following two situations occurs:

1. A query to the oracle is outside the domain of σ.
2. $\widehat{M}_e$ halts.

Since we require each OTM to scan the entire type-0 input string, it follow that such $\beta_{\widehat{M}_e}$ is nontrivial. It is also clear that $\beta_{\widehat{M}_e}$ is strongly $\mathcal{F}$-monotone. Also, since $\widehat{M}_e$ is total on $\mathcal{T} \times \mathbf{N}$, $\beta_{\widehat{M}_e}$ is convergent. Therefore, $\beta_{\widehat{M}_e}$ is a well defined type-2 time bound. Moreover, we can have a locking detector for $\beta_{\widehat{M}_e}$, which simply checks if situation 2 occurs, and hence $\beta_{\widehat{M}_e}$ is locking detectable.

A computable function $t : \mathbf{N} \to \mathbf{N}$ is said to be *time-constructable* if there exists a Turing machine M such that, for each $x \in \mathbf{N}$, the number of steps performed by M on x is exactly the same as the value $t(x)$. An analogous notion of time-constructability can apply to type-2 time bounds. It is clear that any type-2 time bound $\beta_{\widehat{M}_e}$ determined by the OTM $\widehat{M}_e$ is time-constructable. This implies that $\{\beta_{\widehat{M}_e} : e \in \mathbf{N}\}$ is a measured set [Blu67] (see Definition 94 on page 122) and suggests that the complexity class determined by $\beta_{\widehat{M}_e} \in \mathbf{T}_2\mathbf{TB}$ should be honest [MM69] in some sense. However, we will see later that the original motivation for wanting time bounds to be honest does not translate to type-2.

3.3 Type-2 Time Bounds in The Limit

In this section we study a few relations between type-2 time bounds and type-2 functionals. More precisely, we will see that each type-2 time bound determines a limit functional.[3]

Definition 48 (Type-2 time bounds in the limit) *Let $\beta \in \mathbf{T}_2\mathbf{TB}$. Define $F_\beta : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ by*

$$F_\beta = \lambda f, x. \left(\lim_{\sigma \to f} \beta(\sigma, x) \right).$$

■

Some obvious properties of F_β are inherited from the properties of β. These include that F_β is a continuous functional, F_β is total on $\mathcal{T} \times \mathbf{N}$, and that F_β is a limit functional. However, F_β is not necessarily computable by an OTM. The following theorem is a variant of a result in [Rog92] stating that not every limit functional is computable.[4]

Theorem 49 *There exists $\beta \in \mathbf{T}_2\mathbf{TB}$ such that F_β is not a computable type-2 functional that is total on $\mathcal{T} \times \mathbf{N}$.*

Proof: Define $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ by

$$\beta(\sigma, x) = \begin{cases} \widehat{\varphi}_x(\sigma^{\sim 0}, x) + |x| + 1 & \text{if } \widehat{M}_x(\sigma^{\sim 0}, x) \downarrow \text{ in } \|\sigma\| \text{ steps, and all} \\ & \text{queries made are in } \mathsf{dom}(\sigma); \\ |x| + 1 & \text{otherwise.} \end{cases} \tag{15}$$

[3]For more details about limit functionals, see [Rog92], page 364.

[4]The original theorem is due to Poúr-EL [PE60]. In Rogers' terms, the theorem states that not every limit functional corresponds to a recursive functional that is total on $\mathcal{R}$.

Clearly, β is a type-2 time bound. Suppose, by contradiction, that $\widehat{\varphi}_e$ computes F_β. Then, for each $f \in \mathcal{T}$, there exists $\sigma \in \mathcal{F}$ such that $\sigma \subset f$, (σ, e) is a locking sequence for $\widehat{\varphi}_e$, and $M_e(f, e)\downarrow$ in s steps and does not query beyond σ. Thus, for each $\tau \supseteq \sigma$,

$$\widehat{\Phi}_e(\sigma^{\sim 0}, e) = \widehat{\Phi}_e(\tau^{\sim 0}, e) = s \;\text{ and }\; \widehat{\varphi}_e(\sigma^{\sim 0}, e) = \widehat{\varphi}_e(\tau^{\sim 0}, e).$$

But, there exists $\tau \in \mathcal{F}$ such that, $\sigma \subseteq \tau \subset f$, $\mathsf{card}(\tau) = s$, and

$$\widehat{\varphi}_e(\sigma^{\sim 0}, e) = \widehat{\varphi}_e(\tau^{\sim 0}, e) = \widehat{\varphi}_e(\tau^{\sim 0}, e) + |e| + 1.$$

This leads a contradiction. □

Note that the β defined in equation (15) is not a locking-detectable type-2 time bound. Otherwise we can decide a point at which β has converged, and thus there always exists a corresponding computable type-2 functional that is total on $\mathcal{T} \times \mathbf{N}$ and computes F_β. We state this in the following theorem.

Theorem 50 *Given $\beta \in \mathbf{T_2TB}$, if β is a locking-detectable type-2 time bound, then F_β is a computable type-2 functional that is total on $\mathcal{T} \times \mathbf{N}$.*

Proof: Let $\ell : \mathcal{T} \times \mathbf{N} \to \{0, 1\}$ be a computable locking detector for β. Define $F_\beta : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ as

$$F_\beta(f, x) = \beta(f_{[n]}, x),$$

where n is the minimal number such that, $\ell(f_{[n]}) = 1$. Clearly, such n in the above equation can be effectively found. □

Corollary 51 *There exists a type-2 time bound that is not locking detectable.*

Proof: We immediately obtain this from Theorems 49 and 50. □

3.4 Clocked Oracle Turing Machines

In this section we describe a scheme for clocking OTMs with type-2 time bounds under the two OTM cost models. Recall that OTM_A denotes the OTM under the answer-length-cost model and OTM_U denotes the OTM under the unit-cost model. Also, recall the notations $\widehat{\varphi}_e, \widehat{\Phi}_e$, and $\widehat{M}_e$ from Section 1.4. Note that the notations for the unit-cost model are differentiated by a superscript u, e.g., $\widehat{M}^u_e$ and $\widehat{\Phi}^u_e$. Since the two models do not differ on computability, we have that, for every $\widehat{\varphi}$-program e, $\widehat{\varphi}_e = \widehat{\varphi}^u_e$. But when an OTM, $\widehat{M}_e$, clocked by β runs under the two different OTM cost models, in general, we will have two different results, i.e., $\widehat{\varphi}_{e,\beta} \neq \widehat{\varphi}^u_{e,\beta}$ (to be defined momentarily). By convention, we assume that an OTM (under either model) halts if and only if it enters a state labeled "stop", and we also assume that there is no next state for an OTM to apply once it enters a "stop" state.

Definition 52 (Clocked OTM) *Fix an OTM, $\widehat{M}_e$, and a $\beta \in \mathbf{T}_2\mathbf{TB}$. We say that the OTM, $\widehat{M}_e$, is clocked by β if and only if $\widehat{M}_e$ is simulated by the procedure shown in Figure 3. An OTM that performs such a simulation is denoted as $\widehat{M}_{e,\beta}$.* ■

Definition 53 (Functionals Computed by Clocked OTMs) *Let $\beta \in \mathbf{T}_2\mathbf{TB}$.*

1. *Fix any OTM_A, $\widehat{M}_e$. Let $\widehat{\varphi}_{e,\beta} : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be the functional computed by the β-clocked OTM_A, $\widehat{M}_{e,\beta}$.*

2. *Fix any OTM_U, $\widehat{M}^u_e$. Let $\widehat{\varphi}^u_{e,\beta} : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be the functional computed by the β-clocked OTM_U, $\widehat{M}^u_{e,\beta}$.* ■

Note that we use the same clocking scheme for the two OTM cost models. We shall discuss in some details that the two different models strongly affect the results of our clocked simulations, and yield two different type-2 complexity theories in the following.

```
Program for Clocked OTM \widehat{M}_{e,β} :

    input
        f : T, x : N;
    var                                                    /* variables declaration */
        σ : F;
        q, y, expense, budget : N;

    σ ⟵ ∅;                                                 /* initialization */
    expense ⟵ 0;
    budget ⟵ β(σ, x);

    Simulate \widehat{M}_e on (f, x) step by step and upon each step completed do:
        expense ⟵ expense + 1;
        if (expense > budget)
            then output 0 and stop;                                            (⇑)
        if (\widehat{M}_e halts with the output y)
            then output y and stop;                                            (⇓)
        if (the step just simulated completes an oracle query)
            then do
                q ⟵ current query;
                if (q ∈ dom(σ))
                    then go-to next :              /* the query has been asked before */
                σ ⟵ σ ∪ {(q, f(q))};
                budget ⟵ β(σ, x);
            end-do;

next :
    resume the simulation of \widehat{M}_e for the next step;

End program
```

Figure 3: A Clocking Scheme for OTMs

Clocked OTM_A: We first discuss the clocked OTM under the answer-length-cost model. By convention, an OTM_A will read the entire oracle answer right after an oracle query is made. Thus, although we update the budget before the OTM_A being simulated actually reads the oracle answer, the followup simulation will do nothing but to scan the oracle answer with cost of the answer's length. Thus, if the new received budget is not big enough for the OTM_A to read the new oracle answer, the simulation will be terminated before any computation other than scanning the oracle answer can actually start. Thus, the simulation scheme in Figure 3 is straightforward yet appropriate for our purposes.

Fix an OTM_A, $\widehat{M}_e$, and $\beta \in \mathbf{T}_2\mathbf{TB}$. Suppose that we are to run the procedure in Figure 3 on $(f, a) \in \mathcal{T} \times \mathbf{N}$. We call the bound provided by clock β during the course of simulation *budget*. If the simulation has overrun the budget provided by β based on the oracle queries ever made, then the simulation of $\widehat{M}_e$ will be terminated by the line marked ($\Uparrow$). In this case we say that $\widehat{M}_e$ is *clipped down* by β on (f, a). We denote this situation by $\widehat{\varphi}_{e,\beta}(f, a) \Uparrow$. On the other hand, if the simulation procedure reaches the line marked ($\Downarrow$), which means that the simulation of $\widehat{M}_e$ on (f, a) is successfully completed, and we denote this situation by $\widehat{\varphi}_{e,\beta}(f, a) \Downarrow$. Clearly, if $\widehat{\varphi}_{e,\beta}(f, a) \Downarrow$ then $\widehat{\varphi}_{e,\beta}(f, a) = \widehat{\varphi}_e(f, a)$.

Since β as a type-2 time bound must be convergent, it follows that, for every $(f, a) \in \mathcal{T} \times \mathbf{N}$, there is $\sigma \subset f$ such that, $\beta(\sigma, a) \downarrow$. Thus, the *budget* in the clocking scheme will eventually converge to a finite number. It follows that the simulation of $\widehat{M}_e$ on (f, a), either completes or is clipped by the clock. Therefore, for any $\beta \in \mathbf{T}_2\mathbf{TB}$ and $\widehat{M}_e$, $\widehat{\varphi}_{e,\beta}$ is a total computable functional of the type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$.

Clocked OTM_U: We now consider the unit-cost model. Under this cost model, an OTM_U does not have to scan the entire oracle answer returned. An intuitive idea to take advantage of this liberty in our clocking scheme is to have the budget updated based on partial oracle answers that are actually scanned. Thus, a different clocking scheme for this purpose is needed, which shall update the budget according to the scanned portion of the oracle answer bit by bit. However, we are forced to give up this idea by the following difficulty.

Recall that, by our conventions, whenever the oracle returns its answer on the tape, the head of the tape for reading the answer will be automatically relocated to the least significant bit of the newly returned answer (see Section 1.4.2 for details). Thus, every oracle answer will be scanned from the least significant bit. Consider the following $\beta \in \mathbf{T_2TB}$. Let $0 < c$.

$$\beta(\sigma, x) = \begin{cases} \sigma(0) + |x| + c & \text{if } 0 \in \mathsf{dom}(\sigma); \\ |x| + c & \text{otherwise.} \end{cases}$$

Let $(f, x) \in \mathcal{T} \times \mathbf{N}$. Suppose $f(0) = a \times 2^{|k|+1}$ for some $a, k \in \mathbf{N}$. If k is sufficiently large, then no $\mathrm{OTM_U}$, on (f, x), clocked by such β could ever get the budget $f(0) + |x| + c$. This is because that the budget is calculated based on $\sigma(0) = 0$ before the least significant nonzero bit in the oracle answer scanned, and the budget will be used up before that. If the clocking procedure should update the budget based on partial oracle answer, this problem will be encountered generally. Therefore, we think it is proper to let the budget be updated once a query is made, regardless how the $\mathrm{OTM_U}$ scans it. Without tracking the portion of the oracle answer scanned, we are able to use the same clocking scheme shown in Figure 3 for $\mathrm{OTM_U}$s. As a result, the clocked $\mathrm{OTM_U}$ may gain budget by just querying oracle.

Difference between $\widehat{\varphi}_{e,\beta}$ and $\widehat{\varphi}^u_{e,\beta}$: Here is an example that shows $\widehat{\varphi}_{e,\beta} \neq \widehat{\varphi}^u_{e,\beta}$ in general. Consider the functional F defined in Example 8 on page 28. F on $(f, x) \in \mathcal{T} \times \mathbf{N}$, outputs 0 if $f(x)$ is even, 1 otherwise. Under the unit-cost model, there is an OTM$_{\mathrm{U}}$ $\widehat{M}_e^u$ that, on every $(f, x) \in \mathcal{T} \times \mathbf{N}$, computes $F(f, x)$ with complexity $\widehat{\Phi}_e^u(f, x) \leq c(|x| + 1)$, where $c \in \mathbf{N}$. Define $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ by

$$\beta(\sigma, x) = c(|x| + 1). \tag{16}$$

It is clear that $\widehat{\varphi}_{e,\beta}^u = \widehat{\varphi}_e$ because, for every (f, x), $\widehat{M}_e^u$ needs a constant number of steps to check the parity the oracle answer, $f(x)$, and hence $\widehat{\varphi}_{e,\beta}^u(f, x) \Downarrow$. On the other hand, if we choose the answer-length-cost model, every answer from the oracle must be scanned. Thus, every $\widehat{M}_e$ that computes F has complexity at least $\widehat{\Phi}_e(f, x) = c(|x|+|f(x)|+1)$. Therefore, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$ with $f(x)$ sufficiently large, we have $\widehat{\varphi}_{e,\beta}(f, x) \Uparrow$.

Arbitrarily Complex $\widehat{\varphi}$-program: Given any computable $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ and a $\beta \in \mathbf{T_2TB}$, we always can write a "bad" $\widehat{\varphi}$-program for F and the cost of computing the $\widehat{\varphi}$-program is not bounded by β. This is stated in the following theorem.

Theorem 54 *Given any computable $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ and $\beta \in \mathbf{T_2TB}$, there is a $\widehat{\varphi}$-program e for F such that $\widehat{\varphi}_e \neq \widehat{\varphi}_{e,\beta}$.*

Proof: Let $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be computable. By the type-2 S-m-n and recursion theorems stated in Theorems 9 and 10, we can construct a $\widehat{\varphi}$-program e such that, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$,

$$\widehat{\varphi}_e(f, x) = \begin{cases} F(f, x) & \text{if } \widehat{\varphi}_{e,\beta}(f, x) \Uparrow, \\ \widehat{\varphi}_e(f, x) + 1 & \text{otherwise.} \end{cases} \tag{17}$$

It is clear that that $\widehat{\varphi}_e = F$ because the otherwise-case in (17) will never occur. Note that we do not require β to be locking detectable in this theorem. $\square$

3.5 Useful Type-2 Time Bounds

In many cases β is not as powerful as it may look. Even if β is locking detectable, the budget given by β in the limit cannot be always useful. Consider the following example. Define $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ by, for every $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$,

$$\beta(\sigma, x) = \begin{cases} |x| + 1 & \text{if } 0 \notin \mathsf{dom}(\sigma); \\ \sigma(0)^{1000} + |x| + 1 & \text{if } 0 \in \mathsf{dom}(\sigma). \end{cases} \tag{18}$$

Under our conventions, every OTM is required to scan the entire type-0 input at the beginning of its computation. Thus, no OTM can really benefit from the huge budget $\sigma(0)^{1000}$ because the initial budget, $|x| + 1$, will be used up after the type-0 input scanned, and no further computation is allowed. In other words, the budget that can be obtained by querying $f(0)$ is inaccessible, and hence useless.

To successfully query $f(q)$, an $\mathrm{OTM_U}$ needs at least $|q| + 1$ steps to place the query, q, onto the tape, whereas an $\mathrm{OTM_A}$ needs at least $|f(q)| + |q| + 2$ steps to place the query, q, and scan the answer, $|f(q)|$. For every $\sigma \in \mathcal{F}$, let $\|\sigma\|$ as before. We abuse the notation by

$$\|\mathsf{dom}(\sigma)\| = \sum_{i \in \mathsf{dom}(\sigma)} (|i| + 1).$$

Therefore, $\|\sigma\|$ is the minimum possible number of steps an $\mathrm{OTM_A}$ needs for querying the entire σ, and $\|\mathsf{dom}(\sigma)\|$ is the number an $\mathrm{OTM_U}$ needs. We have the following definition:

Definition 55 *Let $\beta \in \mathbf{T_2TB}$ and $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$.*

1. *We say that (σ, a) is **$\boldsymbol{\beta}$-queriable**, if there is an $\mathrm{OTM_U}$ $\widehat{M}^u_e$ such that, the β-clocked $\mathrm{OTM_U}$, $\widehat{M}^u_{e,\beta}$, on every (f, a) with $\sigma \subset f$, can successfully query $\mathsf{dom}(\sigma)$ in some order. We say that (σ, a) is **$\boldsymbol{\beta}$-queriable witnessed** by $\widehat{M}^u_e$.*

2. *We say that (σ, a) is **β-checkable**, if there is an OTM$_{\mathrm{U}}$ $\widehat{M}^u_e$ such that, on every $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{\varphi}^u_{e,\beta}(f, x) \Downarrow$, and*

$$\widehat{\varphi}^u_{e,\beta}(f,x) = \begin{cases} 1 & \text{if } \sigma \subset f \text{ and } x = a; \\ 0 & \text{otherwise.} \end{cases}$$

*We say that (σ, a) is **β-checkable witnessed** by $\widehat{M}^u_e$.* ■

Note that we use the unit-cost model OTM in the above definition. Let $\beta \in \mathbf{T_2TB}$ and $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$. Since every $\beta \in \mathbf{T_2TB}$ must be convergent, it is clear that that not every $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$ is β-checkable or β-queriable.

Let $\beta \in \mathbf{T_2TB}$ and $(\sigma, a) \in \mathcal{T} \times \mathbf{N}$. Suppose that (σ, a) is β-queriable witnessed by $\widehat{M}^u_e$. Although $\widehat{M}^u_{e,\beta}$ can gain budget by simply querying $\mathsf{dom}(\sigma)$, the budget however is based on information of σ. Thus, not for every $\tau \in \mathcal{F}$ with $\mathsf{dom}(\sigma) = \mathsf{dom}(\tau)$, (τ, a) is also β-queriable witnessed by some $\widehat{M}^u$. For a β-queriable (σ, a), we can have an OTM$_{\mathrm{U}}$ clocked by β to print out $\mathsf{dom}(\sigma)$ in some order, but there may be no OTM$_{\mathrm{A}}$ clocked by β to print out $\mathsf{dom}(\sigma)$ in any order for the obvious reasons. If (σ, a) is β-checkable, then σ can be printed in some order by a β-clocked OTM$_{\mathrm{A}}$ on every (f, a) with $\sigma \subset f$.

We further give two properties with which the time bounds are more useful for our purposes in the following chapters.

Definition 56 *Let $\beta \in \mathbf{T_2TB}$.*

1. *We say that β is **accessible** if and only if there is an OTM$_{\mathrm{U}}$ $\widehat{M}^u_e$ such that, all minimal locking fragments of β are β-queriable witnessed by $\widehat{M}^u_e$.*

2. *We say that β is **useful** if and only if there is an OTM$_{\mathrm{U}}$ $\widehat{M}^u_e$ such that, all minimal locking fragments of β are β-checkable witnessed by $\widehat{M}^u_e$.* ■

Clearly, if β is useful, then it is also accessible. Note that a locking detectable type-2 time bound is not necessarily useful or accessible, e.g., the β defined in (18) is locking detectable, but neither useful nor accessible. For the inverse, we have the following theorem.

Theorem 57 *If $\beta \in \mathbf{T_2TB}$ is accessible, then β is strong.*

Proof: Let $\beta \in \mathbf{T_2TB}$ be accessible. Fix any OTM$_\mathrm{U}$ $\widehat{M}_e^u$ that witnesses β being accessible. Clearly, given any $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, we can effectively decide if a given (σ, a) is β-queriable. If not, (σ, a) must be a locking fragment of β. Thus, we can have $\ell : \mathcal{F} \times \mathbf{N} \to \{0, 1\}$ such that, ℓ, on every (σ, x), returns 1 if (σ, x) is not β-queriable, or 0 otherwise. (Note that $\widehat{\varphi}_e = \widehat{\varphi}_e^u$.) $\square$

Theorem 58 *Let $\beta \in \mathbf{T_2TB}$.*

1. *For every $a \in \mathbf{N}$, $(\emptyset, a)$ is β-checkable.*

2. *For every $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$, if (σ, a) is β-checkable, then*

$$\beta(\sigma, a) \geq \|\sigma\| + |a| + 1.$$

$\blacksquare$

Theorem 59 *Let $\beta \in \mathbf{T_2TB}$.*

1. *If β is accessible, then for every minimal locking fragment, (σ, a), of β, we have*

$$\forall \tau \subseteq \sigma \, [\beta(\tau, a) \geq \|\mathsf{dom}(\tau)\| + |a| + 1] \, .$$

2. *If β is useful, then for every minimal locking fragment, (σ, a), of β, we have*

$$\forall \tau \subseteq \sigma \, [\beta(\tau, a) \geq \|\tau\| + |a| + 1] \, .$$

$\blacksquare$

The proofs of the two theorem above are straightforward and omitted. Note that the inverse of each case in Theorem 59 being true requires that the set of the minimal locking fragments of β is recursively enumerable, which in general is not true.

The importance of usefulness is as follows. Let e be a $\widehat{\varphi}$-program. Suppose $\widehat{\varphi}_e = \mathbf{C}(\beta)$. If β is useful, then we can patch e on some (τ, a) finitely often under the budget of β, provided (τ, a) is not a locking fragment of β. We will see the importance of having a β useful in the following chapters. More details about patching a program will be discussed in Theorem 69 and Section 4.2.

Chapter 4

Type-2 Complexity Classes

A complexity class is intuitively a collection of functions each of which is computable with a cost no greater than a particular bound, where this cost is assigned through a particular machine/cost model. This chapter formalizes this intuitive notion through the tools developed in Chapters 2 and 3. There are a few alternatives for defining our type-2 complexity classes. Some of the complexity theorems established in the following chapters are sensitive to the differences between the formulations and others are not. It turns out that none of the alternatives can provide us an exact analog of the familiar type-1 notion of complexity classes. We give reasons for our choice of the definition of type-2 complexity classes and begin our study of the properties of these classes.

4.1 Type-2 Time Complexity Classes

Given $F, G : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$, recall the notation, $X_{[F>G]}$, that denotes the subset of $\mathcal{T} \times \mathbf{N}$ such that, for every $(f, x) \in X_{[F>G]}$, $F(f, x) > G(f, x)$. The same usage applies to $X_{[F=X]}, X_{[F\leq G]}$, and so on. Also, recall from the previous two chapters the properties of a $\beta \in \mathbf{T_2TB}$ (Type-2 Time Bounds), the functional $\widehat{\varphi}_{e,\beta} : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ computed by the β-clocked $\mathrm{OTM_A}$, $\widehat{M}_{e,\beta}$, the functional $\widehat{\varphi}^u_{e,\beta} : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ computed by the β-clocked $\mathrm{OTM_U}$, $\widehat{M}^u_{e,\beta}$, and compact sets in the induced topology determined by some fixed functionals. Given any type-2 time bound β, we shall define a type-2 complexity class that is a collection of computable functionals of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$, and every functional in this complexity class is computed by some β-clocked $\mathrm{OTM_A}$ $\widehat{M}_e$ (or $\mathrm{OTM_U}$ $\widehat{M}^u_e$, when the unit-cost model is concerned) on all inputs but possibly "a few" exceptions. Chapter 2 introduced a class of induced topologies in order to give a formal definition for the intuitive concept of "a few type-1 objects" that roughly corresponds to "finitely many type-0 objects" used in defining type-1 complexity classes. Since we have different choices in defining an induced topology, we have a few alternatives in defining type-2 complexity classes.

Definition 60 (Exception Set, $E_{e,\beta}$) *Let $\beta \in \mathbf{T_2TB}$ and $e \in \mathbf{N}$. Define*

$$E_{e,\beta} = \left\{ (f,x) \in \mathcal{T} \times \mathbf{N} \;\middle|\; \widehat{\varphi}_{e,\beta}(f,x) \Uparrow \right\}.$$

We say that $E_{e,\beta}$ is the exception set of the simulation of $\widehat{M}_e$ clocked by β. ∎

Since $\widehat{\varphi}_{e,\beta}$ is the functional computed by $\widehat{M}_e$ clocked by β, we usually say that $E_{e,\beta}$ is the exception set of $\widehat{\varphi}_{e,\beta}$ for simplicity. Consider the clocking scheme shown in Figure 3 on page 78. Clearly, the computation of $\widehat{M}_e$ on (f,x) is clipped by β if and only if $(f,x) \in E_{e,\beta}$. Similarly, we also define $E^u_{e,\beta}$ to be the exception set of $\widehat{\varphi}^u_{e,\beta}$ in the same way except that the base machine in the clocking scheme in Figure 3 is an $\mathrm{OTM_U}$, $\widehat{M}^u_e$:

Definition 61 (Exception Set, $E^u_{e,\beta}$) *Let $\beta \in \mathbf{T_2TB}$ and $e \in \mathbf{N}$. Define*

$$E^u_{e,\beta} = \left\{ (f,x) \in \mathcal{T} \times \mathbf{N} \;\middle|\; \widehat{\varphi}^u_{e,\beta}(f,x) \Uparrow \right\}.$$

We say that $E^u_{e,\beta}$ is the exception set of the simulation of $\widehat{M}^u_e$ clocked by β. ∎

We are now in a position to define a type-2 complexity class determined by a $\beta \in \mathbf{T_2TB}$. The compactness of the exception set is a natural standard for a "good" clocked simulation. The only problem remains is: Which topology we should choose? We have ruled out the commonly used Baire space topology because the exception set is either empty or noncompact in the topology when computable functionals are considered (see Theorem 25).

4.1.1 Using $\mathbb{T}_{max}(\widehat{\varphi}_e, F_\beta)$

Consider $\mathcal{C}(t)$, the type-1 complexity class determined by $t \in \mathcal{R}$, defined in (14) on page 65. We have that if $f \in \mathcal{C}(t)$, then there is some φ-program e for f such that, $\Phi_e \leq^* t'$, where $t'(x) = t(|x|)$ for every $x \in \mathbf{N}$. Thus, for a type-2 time bound β, we want to define a type-2 complexity class $\mathbf{C}(\beta)$ so that, if $F \in \mathbf{C}(\beta)$, then there is some $\widehat{\varphi}$-program e for F such that, $\widehat{\Phi}_e \leq_2^\square F_\beta$, where F_β is the functional of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$ determined by β (see Section 3.3) and $\leq_2^\square$ is one of the type-2 almost-everywhere relations discussed in Chapter 2. It seems reasonable to consider the induced topology determined by every minimal locking fragment that locks both $\widehat{\varphi}_e$ and F_β. We thus try the topology $\mathbb{T}_{max}(\widehat{\varphi}_e, F_\beta)$ described in Definition 27. Recall Definition 28 on page 55 that, for continuous $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$,

$$F \leq_{\mathbf{2}}^{\oplus} G \text{ if and only if } X_{[F>G]} \text{ is compact in } \mathbb{T}_{max}(F, G).$$

Definition 62 (The Type-2 Time Bound Class, $\mathbf{C}_{max}(\beta)$) *Given $\beta \in \mathbf{T_2TB}$, define*

$$\mathbf{C}_{max}(\beta) = \left\{ \widehat{\varphi}_e : \mathcal{T} \times \mathbf{N} \to \mathbf{N} \;\middle|\; e \in \mathbf{N} \textit{ and } E_{e,\beta} \textit{ is compact in } \mathbb{T}_{max}(\widehat{\varphi}_e, F_\beta) \right\}.$$

■

Let $\beta \in \mathbf{T_2TB}$ and $e \in \mathbf{N}$. Since β is $\mathcal{F}$-monotone, it is clear that, if $E_{e,\beta}$ is compact in $\mathbb{T}_{max}(\widehat{\varphi}_e, F_\beta)$, then $\widehat{\Phi}_e \leq_{\mathbf{2}}^{\oplus} F_\beta$. However, the two sets $E_{e,\beta}$ and $X_{[\widehat{\Phi}_e > F_\beta]}$ are not necessarily equal.

By Theorem 29, $\leq_{\mathbf{2}}^{\oplus}$ is not transitive. We can use the same trick in the proof of Theorem 29 to construct a type-2 computable functional F, and $\beta_1, \beta_2 \in \mathbf{T_2TB}$ such that, for every $\widehat{\varphi}_e$ that computes F we have $\widehat{\Phi}_e \leq_{\mathbf{2}}^{\oplus} F_{\beta_1} \leq_{\mathbf{2}}^{\oplus} F_{\beta_2}$ but $\widehat{\Phi}_e \not\leq_{\mathbf{2}}^{\oplus} F_{\beta_2}$. A more serious problem with $\mathbb{T}_{max}$ is that Definition 62 fails to have a key intuitive property of a complexity class, which is: If we increase the time bound, we should be able to compute at least as much as we could before. Formally, we have the following nuisance result.

Theorem 63 *There exist $\beta_1, \beta_2 \in \mathbf{T_2TB}$ such that,*

$$\beta_1 \leq \beta_2 \text{ and } \mathbf{C}_{max}(\beta_1) \not\subseteq \mathbf{C}_{max}(\beta_2).$$

Proof: Fix $k \in \mathbf{N}$. Consider the following functional $F_k : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$,

$$F_k(f,x) = \begin{cases} 2^k & \text{if } x = 0; \\ 0 & \text{if } x \neq 0. \end{cases}$$

Clearly, F_k is computable. Recall our conventions that $|x| = 1$ when $x = 0$, and each OTM needs $|x| + 1$ steps for scanning the type-0 input at the beginning of its computation. Thus, for any $\widehat{\varphi}$-program e for F_k, we have, for each $(f,x) \in \mathcal{T} \times \mathbf{N}$,

$$\widehat{\Phi}_e(f,x) \;\geq \begin{cases} k+2 & \text{if } x = 0; \\ |x|+2 & \text{if } x \neq 0. \end{cases}$$

Moreover, we can construct a $\widehat{\varphi}$-program e for F_k such that, for each $(f,x) \in \mathcal{T} \times \mathbf{N}$,

$$\widehat{\Phi}_e(f,x) \;\leq \begin{cases} k+c & \text{if } x = 0; \\ |x|+c & \text{if } x \neq 0, \end{cases}$$

where $c \geq 1$ is a constant depending on OTMs but independent of the choice of k. We define $\beta_1, \beta_2 : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ by

$$\beta_1(\sigma, x) = |x| + c. \tag{19}$$

$$\beta_2(\sigma, x) = \begin{cases} |x| + c + 2 & \text{if } 0 \in \mathsf{dom}(\sigma); \\ |x| + c & \text{otherwise.} \end{cases} \tag{20}$$

Clearly, $\beta_1 \leq \beta_2$. With a proper choice of k, e.g., let $k > c$, we can have

$$E_{e,\beta_1} = E_{e,\beta_2} = (\!(\emptyset, 0)\!).$$

Since $(\!(\emptyset, 0)\!)$ is the only basic open set of $\mathbb{T}_{max}(\widehat{\varphi}_e, F_{\beta_1})$ that contains E_{e,β_1}, it follows that E_{e,β_1} is compact in $\mathbb{T}_{max}(\widehat{\varphi}_e, F_{\beta_1})$, and hence $F \in \mathbf{C}_{max}(\beta_1)$. However, E_{e,β_2} is noncompact in $\mathbb{T}_{max}(\widehat{\varphi}_e, F_{\beta_2})$. This is because each basic open set, $(\!(\sigma, x)\!) \in \mathbb{T}_{max}(\widehat{\varphi}_e, F_{\beta_2})$, must have $\mathsf{dom}(\sigma) = \{0\}$. Thus, for example, the following open cover for E_{e,β_2} does not have a finite sub-cover:

$$\left\{ (\!(\sigma_0^0, 0)\!), (\!(\sigma_0^1, 0)\!), (\!(\sigma_0^2, 0)\!), \ldots \right\}, \tag{21}$$

where $\sigma_0^i = \{\langle 0, i \rangle\}, i \in \mathbf{N}$. Therefore, E_{e,β_1} is noncompact in $\mathbb{T}_{max}(\widehat{\varphi}_e, F_{\beta_2})$ and hence $F \notin \mathbf{C}_{max}(\beta_2)$. Note that each $(\!(\sigma_0^i, 0)\!)$ above is a basic open set in $\mathbb{T}_{max}(\widehat{\varphi}_e, F_{\beta_2})$ but not so in $\mathbb{T}_{max}(\widehat{\varphi}_e, F_{\beta_1})$. $\square$

4.1.2 Using $\mathbb{T}_{min}(\widehat{\varphi}_e, F_\beta)$

The example given in the proof of Theorem 63 shows that if we increase the time bound, we may also refine the induced topology. In case that an increase in the time bound does not decrease the size of the exception set, but the finer topology changes the exception set from being compact to noncompact, then we have the undesirable phenomenon noted in the previous theorem. We next consider the topology

$\mathbb{T}_{min}(\widehat{\varphi}_e, F_\beta)$. Our idea is to use a coarser topology in such way to remove the effect of a finer topology determined by larger time bounds.

Recall Definition 35: For continuous $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$,

$$F \leq_{\mathbf{2}}^* G \text{ if and only if } X_{[F>G]} \text{ is compact in } \mathbb{T}_{min}(F, G).$$

Definition 64 (The Type-2 Time Bound Class, $\mathbf{C}_{min}(\beta)$) *Given $\beta \in \mathbf{T}_2\mathbf{TB}$, define*

$$\mathbf{C}_{min}(\beta) = \left\{ \widehat{\varphi}_e : \mathcal{T} \times \mathbf{N} \to \mathbf{N} \;\middle|\; e \in \mathbf{N} \textit{ and } E_{e,\beta} \textit{ is compact in } \mathbb{T}_{min}(\widehat{\varphi}_e, F_\beta) \right\}.$$

■

One can see that the examples and arguments used in Theorem 63 do not work in the topologies defined by $\mathbb{T}_{min}(\cdot)$, simply because the set in (21) is not an open cover for the exceptions set either in $\mathbb{T}_{min}(\widehat{\varphi}_e, F_{\beta_1})$ or $\mathbb{T}_{min}(\widehat{\varphi}_e, F_{\beta_2})$.

Clearly, if $E_{e,\beta}$ is compact in $\mathbb{T}_{min}(\widehat{\varphi}_e, F_\beta)$, then $\widehat{\Phi}_e \leq_{\mathbf{2}}^* F_\beta$. Also, if $\beta_1, \beta_2 \in \mathbf{T}_2\mathbf{TB}$ with $\beta_1 \leq \beta_2$, then $F_{\beta_1} \leq_{\mathbf{2}}^* F_{\beta_2}$. By Theorem 36, $\leq_{\mathbf{2}}^*$ is transitive. Thus, if $\widehat{\Phi}_e \leq_{\mathbf{2}}^* F_{\beta_1}$ and $F_{\beta_1} \leq_{\mathbf{2}}^* F_{\beta_2}$, then $\widehat{\Phi}_e \leq_{\mathbf{2}}^* F_{\beta_2}$. However, $\widehat{\Phi}_e \leq_{\mathbf{2}}^* F_\beta$ still does not imply $\widehat{\varphi}_e \in \mathbf{C}_{min}(\beta)$, and hence $F_{\beta_1} \leq_{\mathbf{2}}^* F_{\beta_2}$ does not imply $\mathbf{C}_{min}(\beta_1) \subseteq \mathbf{C}_{min}(\beta_2)$. We state this in the next theorem.

Theorem 65 *There exist $\beta_1, \beta_2 \in \mathbf{T}_2\mathbf{TB}$ such that,*

$$\beta_1 \leq \beta_2 \textit{ and } \mathbf{C}_{min}(\beta_1) \not\subseteq \mathbf{C}_{min}(\beta_2).$$

Sketch of Proof: Consider the same β_1 and β_2 defined in (19) and (20), respectively. We shall define a functional $F_k : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that, $F_k \in \mathbf{C}_{min}(\beta_1) - \mathbf{C}_{min}(\beta_2)$. For every $(f, x) \in \mathcal{T} \times \mathbf{N}$, define

$$F_k(f, x) = \begin{cases} 2^{k+|f(0)|} & \text{if } x = 0; \\ 0 & \text{if } x \neq 0. \end{cases}$$

Similar to the arguments in the proof of Theorem 63, we argue that, with a proper choice of k, the best OTM, $\widehat{M}_e$, that computes F_k has

$$E_{e,\beta_1} = E_{e,\beta_2} = (\!(\emptyset, 0)\!).$$

Recall $f_{[n]}$ the definition of the initial n-element segment of $f \in \mathcal{T}$ (page 64). Since the minimal locking fragment of β_1 is $(\emptyset, a)$ whereas $\widehat{\varphi}_e$ is $(f_{[1]}, a)$ for every $(f, a) \in \mathcal{T} \times \mathbf{N}$, it follows that the minimal locking fragments of β_1 always dominate in defining the topology $\mathbb{T}_{min}(\widehat{\varphi}_e, F_{\beta_1})$. Thus, $(\!(\emptyset, 0)\!)$ is the only basic open set containing $(\!(\emptyset, 0)\!)$. Therefore, $(\!(\emptyset, 0)\!)$ is compact in $\mathbb{T}_{min}(\widehat{\varphi}_e, F_{\beta_1})$, and hence $F \in \mathbf{C}_{min}(\beta_1)$. On the other hand, $\widehat{\varphi}_e$ and β_2 have the same minimal locking fragment on any inputs, and hence the set in (21) is an open cover without finite sub-cover for E_{e,β_2} in topology $\mathbb{T}_{min}(\widehat{\varphi}_e, F_{\beta_2})$. Thus, E_{e,β_2} is noncompact in $\mathbb{T}_{min}(\widehat{\varphi}_e, F_{\beta_2})$, and hence $F \notin \mathbf{C}_{min}(\beta_2)$. $\square$

4.1.3 The Type-2 Complexity Class: $\mathbf{C}(\beta)$

As we pointed out in the beginning of Section 4.1.1, if F belongs to a type-2 complexity class determined by a $\beta \in \mathbf{T}_2\mathbf{TB}$, we want a certain type-2 almost everywhere less than or equal to relation, $\widehat{\Phi}_e \leq_2^{\square} F_\beta$, to hold for some $\widehat{M}_e$ that computes F. We can directly apply the definitions of type-2 almost-everywhere relations where we let F_β participate in determining the induced topology for the compactness of the exception set. In such a way, for each $\beta \in \mathbf{T}_2\mathbf{TB}$, $\mathbf{C}_{max}(\beta)$ and $\mathbf{C}_{min}(\beta)$ have the following required properties:

$$[F \in \mathbf{C}_{max}(\beta)] \implies \exists e[\widehat{\varphi}_e = F \ \& \ \widehat{\Phi}_e \leq_{\mathbf{2}}^{\oplus} F_\beta]. \tag{22}$$

$$[F \in \mathbf{C}_{min}(\beta)] \implies \exists e[\widehat{\varphi}_e = F \ \& \ \widehat{\Phi}_e \leq_{\mathbf{2}}^{*} F_\beta]. \tag{23}$$

However, the inverse of neither (22) nor (23) holds. Thus, even though $\leq_2^*$ is transitive, our Goal 2 (page 49) fails. That is, a bigger time bound does not necessarily define a complexity class at least as big as one defined by a smaller time bound. We believe there is no simple definition of type-2 complexity classes such that, the inverse of the implication similar to (22) or (23) can hold under any version of type-2 almost-everywhere relations. Thus, we have to use another approach to fulfill our Goal 2. Before we addressing this problem, we make the following observations.

Fix any $\beta \in \mathbf{T}_2\mathbf{TB}$ and $\widehat{\varphi}$-program e.

1. Suppose that (σ, a) is a minimal locking fragment of $\widehat{\varphi}_e$ but not of β. According to the definition of $\mathbb{T}_{min}(\widehat{\varphi}_e, F_\beta)$, $(\!(\sigma, a)\!)$ is a basic open set of $\mathbb{T}_{min}(\widehat{\varphi}_e, F_\beta)$. Moreover, there is no f and g such that, $\sigma \subset f$, $\sigma \subset g$, and $\widehat{\varphi}_{e,\beta}(f, a) \Downarrow$, but $\widehat{\varphi}_{e,\beta}(g, a) \Uparrow$. This is because the computations of $\widehat{M}_e$ on (f, a) and (g, a) are the same. In other words, if β converges after $\widehat{\varphi}_e$ does, the locking fragment of $\widehat{\varphi}_e$ is sufficient to tell the simulation result of $\widehat{\varphi}_{e,\beta}$. Thus, the topology $\mathbb{T}_{min}(\widehat{\varphi}_e, F_\beta)$ is fine enough to reflect the computation of $\widehat{M}_e$ on (f, a) where f is any extension of σ.

2. Suppose that β is locked on (σ, a) but any OTM that computes $\widehat{\varphi}_e$ on (f, a) with $\sigma \subset f$ must query beyond σ. In this case, there *must be* infinitely many[1] extensions of (σ, a) on which $\widehat{M}_e$ will be clipped by β. But the exception set of $\widehat{\varphi}_{e,\beta}$ may still be compact in $\mathbb{T}_{min}(\widehat{\varphi}_e, F_\beta)$, because the locking fragments of β dominate in determining the topology $\mathbb{T}_{min}(\widehat{\varphi}_e, F_\beta)$. Thus in this case, a set being compact in $\mathbb{T}_{min}(\widehat{\varphi}_e, F_\beta)$ does not mean the computations of $\widehat{M}_e$ on the set are finite. Thus, topology $\mathbb{T}_{max}(\widehat{\varphi}_e, F_\beta)$ is a better choice for properly reflecting the computation of $\widehat{M}_e$.

[1]If we consider $\mathrm{OTM_U}$, we should say "In this case, there *may be* infinitely many..."

In both cases, we do not want F_β to play a role in determining the topology. Thus, we come to the following definition of $\mathbf{C}(\beta)$. Recall our conventions that $\mathbb{T}(\cdot) = \mathbb{T}_{min}(\cdot)$.

Definition 66 (The Type-2 Time Bound Class) *Let $\beta \in \mathbf{T}_2\mathbf{TB}$. Define the set of computable type-2 functionals $\mathbf{C}(\beta)$ as*

$$\mathbf{C}(\beta) = \left\{ \widehat{\varphi}_e : \mathcal{T} \times \mathbf{N} \to \mathbf{N} \;\middle|\; e \in \mathbf{N} \textit{ and } E_{e,\beta} \textit{ is compact in } \mathbb{T}(\widehat{\varphi}_e) \right\}.$$

■

One can see that the tricks used in Theorems 63 and 65 do not apply to the complexity classes defined by Definition 66. We have the following theorem indicating that Goal 2 (page 49) is fulfilled.

Theorem 67 *For every $\beta_1, \beta_2 \in \mathbf{T}_2\mathbf{TB}$,*

$$[\beta_1 \leq \beta_2] \implies [\mathbf{C}(\beta_1) \subseteq \mathbf{C}(\beta_2)].$$

Proof: Since the clock itself does not participate in determining the induced topology, it follows that, unless the exception set itself changes, the compactness of the exception set will remain the same with different clocks.

Let $\beta_1, \beta_2 \in \mathbf{T}_2\mathbf{TB}$ with $\beta_1 \leq \beta_2$. It is clear that a bigger type-2 time bound provides possibly more budget at any moment during the course of the computation. Thus, for any $\widehat{M}_e$, $E_{e,\beta_2} \subseteq E_{e,\beta_1}$. By Lemma 34, if E_{e,β_1} is compact in $\mathbb{T}(\widehat{\varphi}_e)$, so is its subset, E_{e,β_2}. It follows that if $\widehat{\varphi}_e \in \mathbf{C}(\beta_1)$, then $\widehat{\varphi}_e \in \mathbf{C}(\beta_2)$. □

Given any $\beta \in \mathbf{T}_2\mathbf{TB}$, we consider $\mathbf{C}(\beta)$ as our standard type-2 complexity class. Thus, by "the type-2 complexity class determined by β", we mean $\mathbf{C}(\beta)$ defined by

Definition 66 for the rest of this thesis. Unlike $\mathbf{C}_{max}(\beta)$ and $\mathbf{C}_{min}(\beta)$, the implication that

$$(\widehat{\varphi}_e \in \mathbf{C}(\beta)) \Rightarrow (\widehat{\Phi}_e \leq_{\mathbf{2}}^{*} F_\beta)$$

is not obvious. We will prove this in the next section.

4.2 Properties of $\mathbf{C}(\beta)$

In this section we study some important properties of type-2 complexity classes defined in the previous section. We first introduce some terminology for later discussion.

Definition 68 *Let $\beta \in \mathbf{T_2TB}$, and $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$.*

1. *If $F \in \mathbf{C}(\beta)$, then we say that the computational complexity of F is **almost everywhere bounded** by β.*

2. *If there is a $\widehat{\varphi}$-program for F with $E_{e,\beta} = \emptyset$, then we say that the computational complexity of F is **everywhere bounded** by β.* ■

4.2.1 The Cost of Patching e on $\mathbb{T}(\widehat{\varphi}_e)$-Compact $E_{e,\beta}$

Here we present some evidence to support the use of a compact exception set in Definition 66. We show that if the computational complexity of F is almost everywhere bounded by β, then we can have a $\widehat{\varphi}$-program for F with computational complexity that is everywhere bounded by β plus some extra cost. Arguably, the extra cost is reasonable under our machine model. Thus, our agenda in this subsection is to know what is the cost to patch a $\widehat{\varphi}$-program e when $E_{e,\beta}$ is compact in $\mathbb{T}(\widehat{\varphi}_e)$.

The two OTM cost models are likely to have two different patching results. It is always dangerous to make unnecessary queries under the answer-length-cost model because the oracle may return a lengthy answer that will cause the budget to be used up in scanning it, whereas this is not a problem under the unit-cost model since any unnecessary query costs only constant time under the unit-cost model. Nevertheless, we point out that the two cost models can permit similar patching results in an alternative topology that will be introduced in Chapter 7.

Recall Definition 55 on page 82 for β-checkable. We notice that β-checkability is the key factor that allows us to patch a program. However, the definition of β-checkable is based on OTMs under the unit-cost model. In this section we need to partially extend the notion of β-checkable to OTMs under the answer-length-cost model. Recall that we use $\mathbf{C_A}(\beta)$ and $\mathbf{C_U}(\beta)$ to denote two complexity classes determined by β under the answer-length-cost model and the unit-cost model, respectively. The next two lemmas concern the answer-length-cost model. For the other cost model, the analogous results immediately follow from Theorem 59.

Lemma 69 *Let $\beta \in \mathbf{T_2TB}$ and $F \in \mathbf{C_A}(\beta)$. Suppose that (σ, x) is a minimal locking fragment of F. Then, there is an $\mathrm{OTM_A}$ $\widehat{M_e}$ such that, (σ, x) is β-checkable as witnessed by $\widehat{M_e}$.*

Sketch of Proof: Let (σ, x) be a minimal locking fragment of F. We need to argue that, for every $f \in \mathcal{T}$, if there is some $i \in \mathsf{dom}(\sigma)$ such that $f(i) \neq \sigma(i)$, β should return enough budget to scan the oracle answer $f(i)$. This is true because the information of f at i is needed for some computation of F; if β fails to provide enough budget for scanning $f(i)$, there will be no $\widehat{\varphi}$-program i for F with $E_{i,\beta}$ compact in $\mathbb{T}(F)$ under answer-length-model. $\square$

The lemma above does not, however, immediately imply that there is an $\mathrm{OTM_A}$ that can check more than one minimal locking fragment in one run with cost bounded by β. The fragments we want to check need to be organized carefully so that the computation does not waste the budget provided by β.

Lemma 70 *Let $\beta \in \mathbf{T}_2\mathbf{TB}$ and $F \in \mathbf{C}_{\mathbf{A}}(\beta)$. Also, let*

$$O = \{(\sigma_0, x_0), (\sigma_1, x_1), \ldots, (\sigma_n, x_n)\}$$

be a finite set of minimal locking fragments of F. Then, there is a $\widehat{\varphi}_e$ such that, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$,

$$\widehat{\varphi}_e(f, x) = \begin{cases} 1 & \textit{if } x = a \textit{ and } \sigma \subset f \textit{ for some } (\sigma, a) \in O; \\ 0 & \textit{otherwise,} \end{cases}$$

and $E_{e,\beta} = \emptyset$, under the answer-length-cost model.

Sketch of Proof: Let O be as described in the lemma above. Let

$$\begin{aligned} A &= \left\{ a \;\middle|\; (\sigma, a) \in O \right\}, \\ X_a &= \left\{ x \;\middle|\; x \in \mathsf{dom}(\sigma) \text{ for some } (\sigma, a) \in O \right\}. \end{aligned}$$

Note that, if (σ, a) and (τ, a) are two locking fragments of a computable functional, then $\mathsf{dom}(\sigma) \cap \mathsf{dom}(\tau) \neq \emptyset$. Thus, for each $a \in A$, we can construct a finite decision tree T_a as follows:

- T_a: Each internal node has a label $x \in X_a$. Each edge leaving an internal node with label x has a label $\sigma(x)$ for some $(\sigma, a) \in O$. We use a special symbol # to label every terminal node. For each $(\sigma, a) \in O$, there is a path leading to a terminal node that represents σ. We require that the nodes in a path of T_a must be distinct.

Since A is finite, for every $x \in \mathbf{N}$, deciding whether $x \in A$ can be done in $|x|+1$ steps. The nontriviality property of β allows a computation for such decision. By Lemma 69, each path representing a σ is β-checkable witnessed by an $\mathrm{OTM_A}$. Thus, there is an $\widehat{M}_e$ such that, under the answer-length-cost model, on every $(f, a) \in \mathcal{T} \times \mathbf{N}$, $\widehat{M}_e$ decides if there is a path in T_a leading to a terminal node that represents $\sigma \subset f$ and the cost of making this decision is bounded by β. □

In the following, we first show that every $F \in \mathbf{C_A}(\beta)$ is computable by an $\mathrm{OTM_A}$ with runtime bounded by $2\beta + c$ on all inputs, where c is a constant in $\mathbf{N}$. A similar argument can be easily obtained for proving a much better result under the unit-cost model.

Theorem 71 *Suppose that $\beta \in \mathbf{T_2TB}$ and $F \in \mathbf{C_A}(\beta)$. Then, there exist $c \in \mathbf{N}$ and a $\widehat{\varphi}$-program e such that, $\widehat{\varphi}_e = F$ and $E_{e,2\beta+c} = \emptyset$.*

Proof: Let $F \in \mathbf{C_A}(\beta)$. Suppose that $\widehat{\varphi}_{e'} = F$ and $E_{e',\beta}$ is compact in $\mathbb{T}(F)$ but $E_{e',\beta} \neq \emptyset$. Fix a finite open cover $\mathcal{O}$ of $E_{e',\beta}$,

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

such that, for each $(\!(\sigma_i, x_i)\!) \in \mathcal{O}$, (σ_i, x_i) is a minimal locking fragment of F. Such finite cover always exists because $E_{e',\beta}$ is compact in $\mathbb{T}(F)$. Construct a $\widehat{\varphi}$-program e that computes the following functional:

For every $(f, x) \in \mathcal{T} \times \mathbf{N}$,

$$\widehat{\varphi}_e(f, x) = \begin{cases} \widehat{\varphi}_{e'}(\sigma_i^{\sim 0}, x_i) & \text{if there exists } (\!(\sigma_i, x_i)\!) \in \mathcal{O} \text{ such that} \\ & \sigma_i \subset f \text{ and } x_i = x; \\ \widehat{\varphi}_{e'}(f, x) & \text{otherwise.} \end{cases} \tag{24}$$

Clearly, $\widehat{\varphi}_e = \widehat{\varphi}_{e'} = F$. By Lemma 70, the condition clause of (24) can be decided with cost bounded by β. Let,

$$c = \mathsf{max}\left\{ \widehat{\Phi}_{e'}(\sigma_i^{\sim 0}, x_i) \;\middle|\; (\!(\sigma_i, x_i)\!) \in \mathcal{O} \right\}.$$

The cost of running $\widehat{\varphi}_{e'}(f, x)$ under the otherwise condition is bounded by β. Therefore, the total cost to implemented the above procedure is bounded by $\beta + \beta + c$, hence $E_{e,2\beta+c} = \emptyset$. □

Under the unit-cost model, we have a much better result: Only an extra constant cost is needed to patch a program. Recall that $\mathbf{C_U}(\beta)$ denotes the complexity class determined by β under the unit-cost model.

Theorem 72 *Suppose that $\beta \in \mathbf{T_2TB}$ and $F \in \mathbf{C_U}(\beta)$. Then, there exist $c \in \mathbf{N}$ and a $\widehat{\varphi}$-program e such that, $\widehat{\varphi}_e = F$ and $E^u_{e,\beta+c} = \emptyset$.*

Sketch of Proof: This theorem is obtained based on the fact that, under the unit-cost model, the condition clause in (24) can be decided in a constant time s. By a straightforward analysis,

$$s = k(\|\sigma_0\| + |x_0| + \|\sigma_1\| + |x_1| + \cdots + \|\sigma_n\| + |x_n|),$$

for some constant $k \in \mathbf{N}$. Let

$$c = s + \mathsf{max}\left\{ \widehat{\Phi}^u_{e'}(\sigma_i^{\sim 0}, x_i) \;\middle|\; (\!(\sigma_i, x_i)\!) \in \mathcal{O} \right\}.$$

Thus, for every $(f, x) \in E^u_{e',\beta}$, we have $\widehat{\Phi}_e(f, x) \leq c$. For every $(f, x) \notin E^u_{e',\beta}$, we need some extra search with cost bounded by $s \leq c$ and compute $\widehat{\varphi}$-program e' with cost bounded by β. Therefore, $E^u_{e,\beta+c} = \emptyset$. □

Clearly, under the answer-length-cost model, doubling the time bound in order to patch a program on a compact set is not reasonable. The result as shown in Theorem

72 under the unit-cost model is more reasonable. One easy way out is to change our topology to one determined by the computations of the concerned $\widehat{\varphi}$-program e instead of the functional $\widehat{\varphi}_e$. That is, if $\widehat{\varphi}_e \in \mathbf{C}(\beta)$ with $E_{e,\beta}$ compact in such topology determined by the computation of e, there are only finitely many different computations of e on $E_{e,\beta}$. Thus, if c is the largest cost of these computations of e on $E_{e,\beta}$, then $E_{e,\beta+c} = \emptyset$ under both cost models. In this thesis we do not consider topologies determined by computations until Chapter 7.

4.2.2 Recursive Enumerability of $\mathbf{C}(\beta)$

For type-1 it is easy to show that the finite invariant closure of a complexity class is *recursively enumerable* [Bor72]. However, not every complexity class itself can be recursively enumerated. When the cost bound function t is very small, the complexity class named by t is unlikely to be recursively enumerable [Bor72, LR70]. On the other hand, if t is big enough to bound all finite support functions[2] 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 ([BH74], 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.

Recall that our $\beta \in \mathbf{T}_2\mathbf{TB}$ must be nontrivial. This property, however, does not suffice for us to have a direct lift from the type-1 result. Our difficulty is that, given $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, we may not be able to check $\sigma \subset f$ for every $f \in \mathcal{T}$, even if $\sigma \subseteq \tau$ for some locking fragment (τ, x) of β under the answer-length-cost model. Thus,

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

intuitively, *usefulness* seems to be a necessary condition for a $\mathbf{C}(\beta)$ being recursively enumerable. However, we do not know, in general, if there is a $\mathbf{C}(\beta)$ that is not recursively enumerable. On the other hand, this problem can be easily solved under the unit-cost model. We partially solve this difficulty in the following.

Recall Definitions 46 and 47 for locking detectable β. Let ℓ be a recursive locking detector for β. In general, the following equivalence does not hold.

$$\ell(\sigma, x) = 1 \iff \beta(\sigma, x) \downarrow .$$

However, the technique used in the next theorem needs to know the exact points at which β converges so we do not patch a program beyond the minimal locking fragments of β. Thus, we require a stronger detectability from β in our construction.

Definition 73 (Strong Locking Detectability) *Let $\beta \in \mathbf{T}_2\mathbf{TB}$. We say that β is* ***strongly locking detectable*** *iff there is a recursive locking detector for β such that, for every $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$,*

$$\ell(\sigma, x) = 1 \iff \beta(\sigma, x) \downarrow .$$

■

Theorem 74 *For every useful and strongly locking detectable $\beta \in \mathbf{T}_2\mathbf{TB}$, $\mathbf{C_A}(\beta)$ is recursively representable.*

Proof: Let $\beta \in \mathbf{T}_2\mathbf{TB}$ be strongly locking detectable. Here we modify the type-1 function in [Odi99] on page 47 and lift it to type-2. In stead of enumerate all finite invariants[3] of $\mathbf{C_A}(\beta)$, we argue that the recursive function g, constructed in the following, provides a recursive presentation of $\mathbf{C_A}(\beta)$.

[3]In fact, we do not have a precise definition of finite invariant for type-2 functionals.

By the S-m-n theorem, there is a recursive function $g : \mathbf{N} \times \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ such that, for every $e, a, b, x \in \mathbf{N}$ and $f \in \mathcal{T}$,

$$\widehat{\varphi}_{g(e,a,b)}(f,x) = \begin{cases} \widehat{\varphi}_e(\tau^{\sim 0}, x) & \text{if } \forall\langle\sigma,y\rangle\,[(\langle\sigma,y\rangle \le a) \Rightarrow \widehat{\varphi}_{e,b}(\sigma^{\sim 0}, y)\Downarrow] \text{ and} \\ & \quad \forall\langle\sigma,y\rangle\,[(a < \langle\sigma,y\rangle \le \langle\tau,x\rangle \ \wedge\ \forall\eta \subseteq \sigma(a < \langle\eta,y\rangle)) \\ & \quad \Rightarrow \widehat{\varphi}_{e,\beta}(\sigma^{\sim 0}, y)\Downarrow]\text{, where } \tau \subset f \text{ and } \langle\tau,x\rangle \text{ is a} \\ & \quad \text{minimal locking fragment of } \beta. \\ 0 & \text{otherwise.} \end{cases} \tag{25}$$

If $F \in \mathbf{C_A}(\beta)$, then there is a $\widehat{\varphi}$-program e for F such that, $E_{e,\beta}$ is compact in $\mathbb{T}(F)$. Thus, if we choose sufficiently large $a, b \in \mathbf{N}$, we have $F = \widehat{\varphi}_{g(e,a,b)}$ for some e. Note that, under answer-length-model, if $\widehat{\varphi}_e \in \mathbf{C_A}(\beta)$ and $E_{e,\beta}$ is compact in $\mathbb{T}(\widehat{\varphi}_e)$, then all queries made outside a locking fragment of β during the course of the computation are not necessary. Thus, if the condition in (25) holds, $\widehat{\varphi}_e(f,x) = \widehat{\varphi}_e(\tau^{\sim 0}, x)$, 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 $\widehat{\varphi}_{g(e,a,b)} \in \mathbf{C_A}(\beta)$. 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., $\widehat{\varphi}$-program $g(e,a,b)$ on every $(f,x) \in S$, the condition of the first clause in (25) 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:

$$\mathcal{O} = \{(\!(\sigma_0, x_0)\!), (\!(\sigma_1, x_1)\!), \ldots, (\!(\sigma_n, x_n)\!)\}, \tag{26}$$

such that, for each $(\!(\sigma_i, x_i)\!) \in \mathcal{O}$, (σ_i, x_i) is a minimal locking fragment of F. By the construction, $\sigma_i \subseteq \tau$ for some minimal locking fragment of β, i.e.,

$\beta(\tau, x_i)\downarrow$. Let $t : \{0, 1, \dots, n\} \to \mathbf{N}$ be a finite function such that, for each $i \le n$, $t(i) = \widehat{\varphi}_e(\sigma_i^{\sim 0}, x_i)$, where $(\!(\sigma_i, x_i)\!) \in \mathcal{O}$. Then, we construct a $\widehat{\varphi}$-program e' as follows:

$$\widehat{\varphi}_{e'}(f, x) = \begin{cases} t(i) & \text{if there exists } (\!(\sigma_i, x_i)\!) \in \mathcal{O} \text{ such that} \\ & \sigma_i \subset f \text{ and } x_i = x; \\ 0 & \text{otherwise.} \end{cases} \tag{27}$$

Clearly, $\widehat{\varphi}_{e'} = F$. Since β is useful and each (σ_i, x_i) is a minimal locking fragment of β or σ_i is a proper subset of some minimal locking fragments of β, it follows that, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, we can check the condition clause in (27) and output 0 or $t(i)$ for some $i \le n$ with cost bounded by β. Therefore, $F \in \mathbf{C}_{\mathbf{A}}(\beta)$ in this case.

Case *(ii)***:** S is not compact in $\mathbb{T}(F)$. Let

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

be any open cover of S such that, for each $(\!(\sigma_i, x_i)\!) \in \mathcal{O}$, (σ_i, x_i) is a minimal locking fragment of F. According to the condition clause in (25), if there are infinitely many (τ, x) such that,

$$\forall\langle\sigma, y\rangle \left[(a < \langle\sigma, y\rangle \le \langle\tau, x\rangle \ \wedge\ \forall\eta \subseteq \sigma(a < \langle\eta, y\rangle)) \Rightarrow \widehat{\varphi}_{e,\beta}(\sigma^{\sim 0}, y) \Downarrow\right] \tag{28}$$

is true, then it follows that, for all but finitely many $\langle\tau, x\rangle$, (28) is true. Thus, $E_{e,\beta}$ is compact. Moreover, 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

$$A = \left\{(f, x) \;\middle|\; F(f, x) \ne \widehat{\varphi}_e(f, x)\right\}$$

is also compact in $\mathbb{T}(F)$. Thus, we can patch e on A and the computational cost of the patched $\widehat{\varphi}$-program on A does not cause it to be removed from $\mathbf{C_A}(\beta)$. Therefore, $F \in \mathbf{C_A}(\beta)$. $\square$

Under the unit-cost model, we do not need β to be useful and strongly locking detectable in order to check the condition in (27) and patch e with cost bounded by β. Thus, we have

Theorem 75 *For every $\beta \in \mathbf{T_2TB}$, $\mathbf{C_U}(\beta)$ is recursively representable.*

Sketch of Proof: Assume that β is locking detectable. We will prove in Section 4.3 that we can make this assumption without loss of generality (see Theorem 88 on page 115). Under the unit-cost model, it is possible that an $F \in \mathbf{C_U}(\beta)$ needs to query the oracle at some places outside the domain of a minimal locking fragment of β. Thus, we modify (25) as follows.

$$\widehat{\varphi}_{g(e,a,b)}(f,x) = \begin{cases} \widehat{\varphi}_e(\tau^{\sim 0}, x) & \text{if } \forall\langle\sigma,y\rangle\,[(\langle\sigma,y\rangle \leq a) \Rightarrow \widehat{\varphi}_{e,b}(\sigma^{\sim 0}, y)\Downarrow] \text{ and} \\ & \quad \forall\langle\sigma,y\rangle\,[(a < \langle\sigma,y\rangle \leq \langle\tau,x\rangle \ \wedge\ \forall\eta \subseteq \sigma(a < \langle\eta,y\rangle)) \\ & \quad \Rightarrow \widehat{\varphi}_{e,\beta}(\sigma^{\sim 0}, y)\Downarrow]\text{, where } \tau = f_{[2^{F_\beta(f,x)}]}, \\ & \\ 0 & \text{otherwise.} \end{cases} \tag{29}$$

Since β is locking detectable, $F_\beta(f,x)$ is computable. We can directly apply the same argument in the proof of Theorem 74 without using the usefulness property of β to prove this theorem. $\square$

Conjecture 76 *There is a $\beta \in \mathbf{T_2TB}$ such that, $\mathbf{C_A}(\beta)$ is not recursively representable.* $\blacksquare$

4.2.3 The Budget in The Limit

Given any $\beta \in \mathbf{T_2TB}$, if $\widehat{\varphi}_e \in \mathbf{C}(\beta)$, we should be able to conclude that $\widehat{\Phi}_e$, the computational complexity of $\widehat{\varphi}_e$, is almost everywhere less than or equal to F_β.

Theorem 77 *Let $\beta \in \mathbf{T_2TB}$. Then, for each $\widehat{\varphi}$-program e, we have*

$$\widehat{\varphi}_e \in \mathbf{C}(\beta) \implies \widehat{\Phi}_e \leq_{\mathbf{2}}^* F_\beta.$$

Proof: Suppose $\widehat{\varphi}_e \in \mathbf{C}(\beta)$. By definition, $E_{e,\beta}$ is compact in $\mathbb{T}(\widehat{\varphi}_e)$. By Lemma 32, we know that $\mathbb{T}(\widehat{\varphi}_e, F_\beta) \subseteq \mathbb{T}(\widehat{\varphi}_e)$, it follows that $E_{e,\beta}$ is also compact in $\mathbb{T}(\widehat{\varphi}_e, F_\beta)$. Thus, $\widehat{\Phi}_e \leq_{\mathbf{2}}^* F_\beta$. □

Thus, if the computational complexity of F is *bounded* by β, then there exist $\widehat{\varphi}_e = F$ and $c \in \mathbf{N}$ such that, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{\Phi}_e(f, x) \leq F_\beta(f, x) + c$.

As we mentioned in Section 4.1.3, in general, $\widehat{\Phi}_e \leq_{\mathbf{2}}^* F_\beta$ does not imply $\widehat{\varphi}_e \in \mathbf{C}(\beta)$. This is because we use the induced topology $\mathbb{T}(\widehat{\varphi}_e, F_\beta)$ for $\widehat{\Phi}_e \leq_{\mathbf{2}}^* F_\beta$, but $\mathbb{T}(\widehat{\varphi}_e)$ and $\mathbb{T}(\widehat{\varphi}_e, F_\beta)$ are not necessarily the same topology. It is possible that $\mathbb{T}(\widehat{\varphi}_e, F_\beta)$ is coarse enough to make $E_{e,\beta}$ compact while $\mathbb{T}(\widehat{\varphi}_e)$ is not. One can check the example shown in the proof of Theorem 65, which is the case that $\mathbb{T}(\widehat{\varphi}_e) \subset \mathbb{T}(\widehat{\varphi}_e, F_\beta)$ and $E_{e,\beta}$ is compact in $\mathbb{T}(\widehat{\varphi}_e)$ but not in $\mathbb{T}(\widehat{\varphi}_e, F_\beta)$. However, even if we have $\mathbb{T}(\widehat{\varphi}_e) = \mathbb{T}(\widehat{\varphi}_e, F_\beta)$ for particular β and $\widehat{\varphi}_e$, in general $\widehat{\Phi}_e \leq_{\mathbf{2}}^* F_\beta$ still does not imply $\widehat{\varphi}_e \in \mathbf{C}(\beta)$. To see this, consider the course of a particular computation. Since the queries must be posed sequentially, we can construct a type-2 time bound that holds the budget until some required queries have been made. This does not change the budget in the limit but the clocked OTM may be clipped by the clock during the period of collecting the necessary queries. We state this in the following theorem.

Theorem 78 *There exist $\beta \in \mathbf{T_2TB}$ and a computable $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that, (i) $F \notin \mathbf{C}(\beta)$, (ii) there is $\widehat{M}_e$ that computes F such that, $\mathbb{T}(\widehat{\varphi}_e) = \mathbb{T}(\widehat{\varphi}_e, F_\beta)$ and $\widehat{\Phi}_e \leq_{\mathbf{2}}^* F_\beta$.*

Proof: Define $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ by

$$F(f, x) = f(f(0)). \tag{30}$$

Clearly, such F is computable, and any program for F must carry out the following four major actions: (i) scan the type-0 input, (ii) copy 0 to the oracle-query tape and query "$f(0) = ?$", (iii) copy $f(0)$ to the oracle-query tape and query "$f(f(0)) =?$", and (iv) copy $f(f(0))$ to the output tape and stop. Thus, the cost of any OTM, $\widehat{M}_e$, for F must have the lower bound as:

$$|x| + |f(0)| + |f(f(0))| < \widehat{\Phi}_e(f, x).$$

For upper bound, there is a constant $c \in \mathbf{N}$ such that, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$,

$$\widehat{\Phi}_e(f, x) \leq |x| + |f(0)| + |f(f(0))| + c.$$

Define $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ by

$$\beta(\sigma, x) = \begin{cases} |x| + |\sigma(0)| + |\sigma(\sigma(0))| + c & \text{if } \{0, \sigma(0)\} \subseteq \mathsf{dom}(\sigma); \\ |x| + 1 & \text{otherwise.} \end{cases} \tag{31}$$

It is easily seen that $\mathbb{T}(\widehat{\varphi}_e) = \mathbb{T}(\widehat{\varphi}_e, F_\beta)$ because the minimal locking fragments of $\widehat{\varphi}_e$ and β are the same. Since, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{\Phi}_e(f, x) \leq F_\beta(f, x)$, it follows that $\widehat{\Phi}_e \leq_{\mathbf{2}}^{*} F_\beta$. Now, suppose p is an arbitrary $\widehat{\varphi}$-program for F. It is clear that for a sufficiently large value of $f(0)$, $\widehat{M}_p$ on (f, x) when clocked by β cannot make its second query, "$f(f(0)) =?$". Thus, the exception set $E_{p,\beta}$ is noncompact in $\mathbb{T}(\widehat{\varphi}_p)$, and hence $F \notin \mathbf{C}(\beta)$. $\square$

Note that Theorem 78 holds under both models, OTM_A and OTM_U, since the entire value of $f(0)$ has to be scanned in order to make the second query.

The following corollary is obtained by modifying the β in (31) without changing its value in the limit. The modified time bound shall provide enough budget during the course of computing F defined in (30).

Corollary 79 *Given* $\beta, \beta' \in \mathbf{T}_2\mathbf{TB}$,

$$[F_\beta = F_{\beta'}] \not\Rightarrow [\mathbf{C}(\beta) = \mathbf{C}(\beta')].$$

Proof: Let β be as defined by (31), and define $\beta' : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ by

$$\beta'(\sigma, x) = \begin{cases} |\sigma(0)| + |\sigma(\sigma(0))| + |x| + c & \text{if } \{0, \sigma(0)\} \subseteq \mathsf{dom}(\sigma); \\ |\sigma(0)| + |x| + c & \text{if } 0 \in \mathsf{dom}(\sigma); \\ |x| + c & \text{otherwise,} \end{cases}$$

where c is the same constant used in (31). Clearly, $F_\beta = F_{\beta'}$. Consider F defined in (30). We know that $F \notin \mathbf{C}(\beta)$. To see $F \in \mathbf{C}(\beta')$, consider an OTM, $\widehat{M}_e$, that computes F. When $\widehat{M}_e$ is clocked by β', once $\widehat{M}_e$ makes the query "$f(0) =?$", β' will update the budget so that $\widehat{M}_e$ is able to make the next query "$f(f(0)) =?$", and finish its computation. Thus, $E_{e,\beta'} = \emptyset$, and hence $F \in \mathbf{C}(\beta')$. $\square$

Corollary 79 suggests that, if we look only at the behavior of a β in the limit, we loose too much information about the way β controls the budget for a computation clocked by β. The budget from β may not be sufficient for some computations, although the limit budget provided by β may be enough. Note that the β constructed in the proof for Theorem 78 is strongly $\mathcal{F}$-monotone. Thus, although strong $\mathcal{F}$-monotonicity can guarantee that the budget will not be shrunk during the course of the computation, it does not guarantee that the budget in the limit can be used in any specific moment of the computation. We may want to have a certain computable operation on β's to let them provide the budget earlier so all computations with complexity bounded by F_β can be prolonged and finish. The following theorem shows that it is impossible.

Theorem 80 *There is no recursive operator* $\Theta : \mathbf{T}_2\mathbf{TB} \to \mathbf{T}_2\mathbf{TB}$ *such that, for any* $\beta \in \mathbf{T}_2\mathbf{TB}$, $\widehat{\varphi}$*-program* e*, and* (f, x) *we have*

$$\widehat{\Phi}_e(f, x) \leq F_\beta(f, x) \iff \widehat{\varphi}_{e,\Theta(\beta)}(f, x) \Downarrow .$$

Proof: Define $F : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$ by

$$F(f, x) = \varphi_x(x).$$

Clearly, we can have a $\widehat{\varphi}$-program e for F such that, for each $(f, x) \in \mathcal{T} \times \mathbf{N}$, if $\varphi_x(x)\downarrow$, then

$$\widehat{\Phi}_e(f, x) \leq \Phi_x(x) + |x| + c,$$

where $c \geq 1$ is some constant depending on the OTM. Define $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ by

$$\beta(\sigma, k) = \begin{cases} t + |k| + c & \text{if } i \text{ is the least element of } \mathsf{dom}(\sigma), \text{ provided:} \\ & \{0, 1, \ldots, i\} \subseteq \mathsf{dom}(\sigma) \text{ and } \sigma(i) = \langle k, t\rangle \text{ for some } t; \\ & \\ |k| + c & \text{if no such } i \text{ exists.} \end{cases}$$

One can check that β is $\mathcal{F}$-monotone, convergent, and nontrivial. Hence $\beta \in \mathbf{T}_2\mathbf{TB}$. Let $K = \{x \mid x \in \mathbf{N}, \varphi_x(x)\downarrow\}$ and $k_0, k_1, \ldots$ be a fixed recursive enumeration of K without repetition. For each $i \in \mathbf{N}$, let $t_i = \Phi_{k_i}(k_i)$. Define $\kappa : \mathbf{N} \to \mathbf{N}$ by

$$\kappa(i) = \langle k_i, t_i\rangle. \tag{32}$$

Clearly, κ is total recursive. For any $k \in K$, there exists $i \in \mathbf{N}$ such that, $k = k_i$ and $\kappa(i) = \langle k, \Phi_k(k)\rangle$, and thus,

$$F_\beta(\kappa, k) = \Phi_k(k) + |k| + c.$$

When $k \in K$, we have

$$\widehat{\Phi}_e(\kappa, k) \leq \Phi_k(k) + |k| + c = F_\beta(\kappa, k).$$

By contradiction, suppose a Θ such as in the statement of the theorem exists. From the definition of F, the OTM $\widehat{M}_e$ would never make any oracle query. Therefore, the overall budget provided by $\Theta(\beta)$ for $\widehat{M}_e$ on (κ, k) is $\Theta(\beta)(\emptyset, k)$. From the assumption, if $x \in K$, then the budget $\Theta(\beta)(\emptyset, x)$ must be enough for computing $\widehat{\varphi}_e(\kappa, x)$. Therefore, we can have a type-1 recursive function to decide if any given x is in K as follows.

$$\varphi_p(x) = \begin{cases} 1 & \text{if } \Phi_x(x) \leq \Theta(\beta)(\emptyset, x), \\ 0 & \text{otherwise.} \end{cases}$$

That gives a recursive characteristic function of K, and hence a contradiction. $\square$

Note that the F constructed in the previous theorem is not total. We can modify this and have the following corollary.

Corollary 81 *There is no recursive operator* $\Theta : \mathbf{T_2TB} \to \mathbf{T_2TB}$ *such that, for any* $\beta \in \mathbf{T_2TB}$ *and* $\widehat{\varphi}_e : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$*, we have that, for every* $(f, x) \in \mathcal{T} \times \mathbf{N}$*,*

$$\widehat{\Phi}_e(f, x) \leq F_\beta(f, x) \iff \widehat{\varphi}_{e,\Theta(\beta)}(f, x) \Downarrow .$$

Sketch of Proof: Define $F : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$ by

$$F(f, x) = \begin{cases} \varphi_x(x) & \text{if } M_x(x) \text{ halts in } f(0) \text{ steps}, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, such F is total computable. Let κ be the same function defined in (32). Fix any $s \in \mathbf{N}$. Define $\kappa^s : \mathbf{N} \to \mathbf{N}$ by

$$\kappa^s(i) = \begin{cases} s & \text{if } i = 0; \\ \kappa(i-1) & \text{otherwise.} \end{cases}$$

Also, define $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ by

$$\beta(\sigma, k) = \begin{cases} \sigma(0) + t + |k| + c & \text{if } i \text{ is the least element of } \mathsf{dom}(\sigma), \text{ provided:} \\ & i \neq 0, \{0, 1, \ldots, i\} \subseteq \mathsf{dom}(\sigma), \text{ and } \sigma(i) = \langle k, t \rangle \\ & \text{for some } t; \\ & \\ |k| + c & \text{if no such } i \text{ exists.} \end{cases}$$

One can see that if $k \in K$, then $F_\beta(\kappa^s, k) = s + \Phi_k(k) + |k| + c$, and, for any $\widehat{\varphi}_e$ for F, we have

$$\widehat{\Phi}_e(\kappa^s, k) \leq \mathsf{min}(s, \Phi_k(k)) + |k| + c \leq F_\beta(\kappa^s, k).$$

Similar to the proof of Theorem 80, if a Θ such as in the statement of the theorem exists, then we can construct a recursive characteristic function of K. $\square$

4.2.4 Arbitrarily Complex Type-2 Classes

There is no time-bound function t that is big enough so that the type-1 complexity class, $DTIME(t)$, contains all recursive functions. We shall show that the analogous type-2 statement is also true. Recall that $f_{[n]}$ is the initial n-element fragment of the total function f and Definition 46 for locking detectors. Let $\beta \in \mathbf{T_2TB}$ be locking detectable and $\ell : \mathcal{F} \times \mathbf{N} \to \{0, 1\}$ be a locking detector of it. We can simply define a functional $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ by

$$F(f, x) = 2^{\beta(f_{[n]}, x)+1},$$

where $n \in \mathbf{N}$ is the least number such that, $\ell(f_{[n]}, x) = 1$. Since ℓ is recursive, such F is computable and total. It is clear that the computation of F cannot be bounded by β. We will see in the next section that no generality is lost by assuming that β is locking detectable.

The outputs of the F constructed above are huge. However, a functional that is difficult to compute is not necessarily large. In type-1 complexity theory, Rabin [Rab60, HH71, Cal88] proved by a cancellation argument that, for every recursive function t, there exists a recursive 0-1 valued function f such that,

$$(\forall e : \varphi_e = f)[t \leq^* \Phi_e].$$

```
Program for F
    input f : T; x : N;
    Q ⟵ ∅;

/* The following loop assures that: For a fixed f, each index should be
canceled at most once. */
    for w = 0 to x do
        S_w ⟵ { k | k ≤ w, k ∉ Q, φ̂_{k,β}(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 */
    return 1 ∸ φ̂_{e,β}(f, x);
End program
```

Figure 4: A Type-2 Cancellation Algorithm

In particular, Rabin's theorem implies 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.

Theorem 82 *For any $\beta \in \mathbf{T_2TB}$, there is a 0-1 valued computable $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that $F \notin \mathbf{C}(\beta)$.*

Proof: Fix any $\beta \in \mathbf{T_2TB}$. Consider the type-2 functional F computed by the program shown in Figure 4. Note that, on any $(f, x) \in \mathcal{T} \times \mathbf{N}$, the Q in the algorithm is finite.

Since, for every $k \in \mathbf{N}$, $\widehat{\varphi}_{k,\beta}$ is a total computable functional, it follows that each S_w in the loop of the algorithm can be obtained effectively. Thus, the loop always terminates. Hence F is a total functional. Clearly, F is 0-1 valued.

Fix an e such that, $\widehat{\varphi}_e \in \mathbf{C}(\beta)$ and $E_{e,\beta}$ is compact in $\mathbb{T}(\widehat{\varphi}_e)$. We shall show that the $\widehat{\varphi}$-program e does not compute F. Since $E_{e,\beta}$ is compact in $\mathbb{T}(\widehat{\varphi}_e)$, there is an

$a \geq e$ such that,

$$(\forall x \geq a)(\forall f \in \mathcal{T})\ [\widehat{\varphi}_{e,\beta}(f,x) \Downarrow]. \tag{33}$$

Fix an $f \in \mathcal{T}$. Suppose that x_0 is the least number such that, $e \leq x_0$ and $\widehat{\varphi}_{e,\beta}(f,x_0) \Downarrow$. Consider the computation of F on (f,x_0). According to the algorithm, e will be in S_{x_0} after the loop exits. If e is the least number in S_{x_0}, then $F(f,x_0) = 1 \dot{-} \widehat{\varphi}_{e,\beta}(f,x_0)$. Since $\widehat{\varphi}_{e,\beta}(f,x_0) \Downarrow$, it follows that $\widehat{\varphi}_{e,\beta}(f,x_0) = \widehat{\varphi}_{e}(f,x_0)$ and hence $F(f,x_0) \neq \widehat{\varphi}_e(f,x_0)$. If e is not the least number in S_{x_0}, we shall prove that for some $x \geq x_0$, $e = \mathsf{min}(S_x)$ and $\widehat{\varphi}_{e,\beta}(f,x) \Downarrow$.

Suppose that $k = \mathsf{min}(S_{x_0})$ and $k \neq e$. By the design of the for-loop in the algorithm, for any $x > x_0$, $k \notin S_x$, because, during 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 (33), if x is sufficiently large, then $\widehat{\varphi}_{e,\beta}(f,x) \Downarrow$. 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, when apply the algorithm to (f,x), $e = \mathsf{min}(S_x)$ and $F(f,x) = 1 \dot{-} \widehat{\varphi}_{e,\beta}(f,x)$. Therefore, $F \neq \widehat{\varphi}_e$. $\square$

4.3 Equivalent Type-2 Time Bounds

We have illustrated that, in general, we cannot expect all type-2 time bounds to come with some desirable properties. For example, locking detectability is a very convenient property, but Corollary 51 asserts that not every type-2 time bound is locking detectable.

Clearly, in order to gain a new budget, we have to make a new query under the cost of the present budget. Thus, if the query needed for gaining the new budget is too big, then the new budget is in fact useless. Recall the definitions of *β-queriable* and *accessible* type-2 time bounds from Definitions 55 and 56, respectively. For any $\beta \in \mathbf{T_2TB}$, if its value changes at a point that is not *β-queriable*, then we can ignore the change, for no computation can really use the new budget. In other words, we can force the β to converge if we did not see any change under the "apparent horizon" based on the current budget.

In this section we consider type-2 time bounds that are not locking detectable and show that the undetectability of any such type-2 time bound must reside somewhere beyond its accessible points. Moreover, we construct an effective operator that, on any $\beta \in \mathbf{T_2TB}$, outputs a locking detectable $\mathbf{T_2TB}$ that determines the same complexity class as $\mathbf{C}(\beta)$.

Definition 83 *Let $\beta, \alpha \in \mathbf{T_2TB}$. We say that β and α are equivalent if and only if the two determine the same complexity class, i.e., $\mathbf{C}(\beta) = \mathbf{C}(\alpha)$.* ■

Definition 84 (Inaccessible Points) *Let $\beta \in \mathbf{T_2TB}$ and $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$ with $\sigma \neq \emptyset$. Suppose that $\mathsf{dom}(\sigma) = \{x_0, x_1, x_2, \ldots, x_n\}$ and $x_0 < x_1 < \cdots < x_n$. For each $x_i \in \mathsf{dom}(\sigma)$, we say that x_i is an* ***inaccessible point*** *in (σ, a) with respect to β if and only if*

1. *$i \neq 0$ and x_{i-1} is an* ***inaccessible point*** *in (σ, a) with respect to β, or*

2. *$|x_i| > \beta(\sigma', a)$, where $\sigma' \subset \sigma$ and $\mathsf{dom}(\sigma') = \mathsf{dom}(\sigma) - \{x_i, x_{i+1}, \ldots, x_n\}$.* ■

Definition 85 *Let $\beta \in \mathbf{T_2TB}$ and $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$. Let $\mathsf{IA}_\beta(\sigma, a)$ denote the set of all inaccessible points in (σ, a) with respect to β.* ■

Clearly, for each $\beta \in \mathbf{T_2TB}$ and $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$, $\mathsf{IA}_\beta(\sigma, a)$ is recursively computable. It is clear that if there are inaccessible points in (σ, a) with respect to β, then (σ, a) is not β-queriable. So, intuitively, to obtain an equivalent locking detectable type-2 time bound from any arbitrary $\beta \in \mathbf{T_2TB}$ we can do the following: On any $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$, we remove $\mathsf{IA}_\beta(\sigma, a)$ from σ resulting in (σ', a); then use $\beta(\sigma', a)$ as the value of the new type-2 time bound. However, we do not require an OTM to make queries in any particular order; hence having inaccessible points in (σ, a) does not mean that, for any $\tau \supset \sigma$, (τ, a) is too not β-queriable. Thus, we face a difficulty in deciding the correct moment to lock our new type-2 time bound. We solve this problem by searching a *base segment*.

Definition 86 (Base Segments) *Let $\beta \in \mathbf{T_2TB}$ and $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$. For $\tau \subseteq \sigma$, we say that (τ, a) is the* ***base segment*** *in (σ, a) with respect to β if and only if*

1. $\mathsf{dom}(\tau)$ *is a finite initial segment of* $\mathbf{N}$,
2. $\mathsf{IA}_\beta(\tau, a) = \emptyset$, *and*
3. *for all* $\tau' \supset \tau$, $\mathsf{IA}_\beta(\tau', a) \neq \emptyset$. ■

It is clear that, given any $\beta \in \mathbf{T_2TB}$ and $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$, we can effectively decide whether the base segment in (σ, a) exists. Moreover, if the base segment does exist in some (σ, a), then we can effectively obtain it. We state this obvious fact in:

Lemma 87 *Let $\beta \in \mathbf{T_2TB}$. Fix any $(f, a) \in \mathcal{T} \times \mathbf{N}$. There exists a unique $\tau \in \mathcal{F}$ with $\tau \subset f$ such that, (τ, a) is the base segment in (σ, a) with respect to β for all $\sigma \in \mathcal{F}$ with $\tau \subseteq \sigma$. Moreover, if $\tau \subset \sigma$, then (σ, a) is not β-queriable.* ■

The above lemma is obvious due to the fact that every $\beta \in \mathbf{T_2TB}$ is convergent. The following theorem is based on the simple idea: Given any $\beta \in \mathbf{T_2TB}$ and $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$, if the base segment (τ, a) in (σ, a) is found, we then lock the type-2 time bound with value $\beta(\tau, a)$.

```
Program for Θ_L(β)

    input σ : F, x : N;
    var τ : F;

    find (τ, x) the base segment in (σ, x) with respect to β;
    if such (τ, x) does not exist
       then τ ⟵ {⟨i, σ(i)⟩ | i ∈ dom(σ) and i ∉ IA_β(σ, x)};
    output β(τ, x);

End program
```

Figure 5: Program for $\Theta_L(\beta)$

Theorem 88 *There is an effective operator* $\Theta_L : \mathbf{T}_2\mathbf{TB} \to \mathbf{T}_2\mathbf{TB}$ *such that, for every* $\beta \in \mathbf{T}_2\mathbf{TB}$*,* $\Theta_L(\beta)$ *is a locking detectable type-2 time bound equivalent to* β*.*

Proof: We construct a $\Theta_L : \mathbf{T}_2\mathbf{TB} \to \mathbf{T}_2\mathbf{TB}$ as follows. Θ_L takes a $\beta \in \mathbf{T}_2\mathbf{TB}$ as its input and outputs the program, $\Theta_L(\beta)$, shown in Figure 5.

It is clear that Θ_L is a recursive operator. Fix a $\beta \in \mathbf{T}_2\mathbf{TB}$. Suppose $\alpha = \Theta_L(\beta)$. We shall prove the following two claims.

Claim 1: $\alpha \in \mathbf{T}_2\mathbf{TB}$ and is locking detectable.

Let $(f, a) \in \mathcal{T} \times \mathbf{N}$.

1. Computability: Since β and IA_β are computable, it follows that the construction shown in Figure 5 is effective.

2. Nontriviality: The value of α is based on the value of β on the same type-0 input. Thus, α is nontrivial as long as β is.

3. $\mathcal{F}$-monotonicity: Since every inaccessible points in any $\sigma \subset f$ will be removed, it follows that, for every $\sigma \subset f$,

$$\alpha(\sigma, a) \leq \lim_{\tau \to f} \alpha(\tau, a).$$

4. Convergence: Since β itself must be convergent, it is clear that if we keep increasing the finite initial segment $f_{[n]}$ and feed $(f_{[n]}, a)$ to α, eventually $(f_{[n]}, a)$ will contain the *base segment*, and then α will be locked.

5. Locking detectability: Since we can effectively decide whether the base segment in (σ, a) exists, by Lemma 87, we can construct a locking detector for α.

Claim 2: $\mathbf{C}(\beta) = \mathbf{C}(\alpha)$.

It is sufficient to prove that, for any $\widehat{\varphi}$-program e and $(f, a) \in \mathcal{T} \times \mathbf{N}$, we have that

$$\widehat{\varphi}_{e,\beta}(f, a) \Downarrow \quad \text{if and only if} \quad \widehat{\varphi}_{e,\alpha}(f, a) \Downarrow . \tag{34}$$

Fix a $\widehat{\varphi}$-program, e, and let $(f, a) \in \mathcal{T} \times \mathbf{N}$. Consider the following two cases.

1. Suppose $\widehat{\varphi}_{e,\beta}(f, a) \Uparrow$. From the construction of α, we know that $\alpha \leq \beta$. Therefore,

$$\widehat{\varphi}_{e,\beta}(f, a) \Uparrow \;\Rightarrow\; \widehat{\varphi}_{e,\alpha}(f, a) \Uparrow .$$

2. Suppose $\widehat{\varphi}_{e,\beta}(f, a) \Downarrow$. Let $\mathsf{dom}(\sigma_t)$ be the collection of queries made during the period from the beginning of the computation up to the moment t. It is clear that there must not be any inaccessible point in (σ_t, a), because otherwise $\widehat{\varphi}_{e,\beta}(f, a) \Uparrow$. Thus, according to the construction of α, we know that $\beta(\sigma_t, a) = \alpha(\sigma_t, a)$. Therefore, at any moment t during the computation, the budget $\alpha(\sigma_t, x)$ is also enough to support the computation. This proves that $\widehat{\varphi}_{e,\alpha}(f, x) \Downarrow$. $\square$

Recall that, if a $\beta \in \mathbf{T_2TB}$ is strongly $\mathcal{F}$-monotone, then the budget will not shrink during the course of any β-clocked computation. Such property is desired in some applications in which we do not want that making oracle queries will cause any penalty to the computation. Thus, we want the operator for Theorem 88 can reserver strong $\mathcal{F}$-monotonicity for every strong β, i.e., if β is strongly $\mathcal{F}$-monotone, then so is $\Theta_L(\beta)$. In fact, the operation constructed in the proof of Theorem 88 does have this property.

Corollary 89 *Let $\Theta_L : \mathbf{T_2TB} \to \mathbf{T_2TB}$ be the effective operator constructed in the proof of Theorem 88. If β is a strong type-2 time bound, then so is $\Theta_L(\beta)$.*

Proof: Let $\beta \in \mathbf{T_2TB}$ and $\Theta_L(\beta) = \alpha$. By Theorem 88, $\alpha \in \mathbf{T_2TB}$. We shall prove that if β is strongly $\mathcal{F}$-monotone, then α is strongly $\mathcal{F}$-monotone too.

Suppose β is strongly $\mathcal{F}$-monotone. Fix any $a \in \mathbf{N}$ and $\sigma_1, \sigma_2 \in \mathcal{F}$ with $\sigma_1 \subseteq \sigma_2$. Since β is strong, we have $\beta(\sigma_1, a) \le \beta(\sigma_2, a)$.

1. Suppose that the base segment, (τ, a), is found in (σ_1, a). By Lemma 87, the same base segment will also be found in (σ_2, a). Thus, $\alpha(\sigma_1, a) = \alpha(\sigma_2, a) = \beta(\tau, a)$.

2. Suppose that the base segment, (τ, a), is found in (σ_2, a) but not found in (σ_1, a). Since the base segment is the maximal segment without inaccessible points, thus $[\mathsf{dom}(\sigma_1) - \mathsf{IA}_\beta(\sigma_1, a)] \subset \mathsf{dom}(\tau)$.

3. Suppose that neither (σ_1, a) nor (σ_2, a) contains the base segment. In this case, we have that $[\mathsf{dom}(\sigma_1) - \mathsf{IA}_\beta(\sigma_1, a)] \subseteq [\mathsf{dom}(\sigma_2) - \mathsf{IA}_\beta(\sigma_2, a)]$.

Since β is strong, it follows that $\alpha(\sigma_1, a) \le \alpha(\sigma_2, a)$ in both cases 2 and 3 above. Therefore, each of the three cases above concludes that $\alpha(\sigma_1, a) \le \alpha(\sigma_2, a)$, and hence α is strong $\mathcal{F}$-monotone. $\square$

With Theorem 88 and Corollary 89 at hand, we henceforth assume that any type-2 time bound is locking detectable without loss of generality.

Chapter 5

Computational Complexity Theory

Computational Complexity Theory has been a very active research area in the past few decades. In this chapter we review some of the most basic results of type-1 complexity theory. In the following chapter we shall be concerned with how these type-1 results translate, or fail to translate, to our type-2 setting.

We begin with a review of Blum's approach [Blu67] to the machine-independent study for the complexity theory. Then, we consider the *Union Theorem, Gap Theorem, Compression Theorem, Honesty Theorem*, and *Speed-up Theorem.* We state these theorems without proofs which can be found in many standard complexity texts [BH74, HU79, WW85, Cal88, Pap94, Odi99]. We also describe the central theoretical issues addressed by the theorems.

5.1 Blum's Axioms and Complexity Measures

Blum in [Blu67] used two axioms to specify two basic requirements that a reasonable dynamic complexity measure should meet.

Definition 90 (Blum Measures [Blu67]) *Let $\langle \psi_i \rangle_{i \in \mathbf{N}}$ be any acceptable programming system. We say that $\langle \Psi_i \rangle_{i \in \mathbf{N}}$ is a **Blum complexity measure** associated with $\langle \psi_i \rangle_{i \in \mathbf{N}}$ if the following two axioms are satisfied:*

> **Axiom 1:** *For all $i, x \in \mathbf{N}$, $\psi_i(x)\downarrow$ if and only if $\Psi_i(x)\downarrow$.*
>
> **Axiom 2:** *There is an effective procedure that, given $i, x, m \in \mathbf{N}$, decides whether $\Psi_i(x) = m$.*

■

Clearly, the number of steps of a TM performs is a Blum measure. The naive space measure of simply counting the number of squares used in a TM computation does not quite work as a Blum measure since a machine can loop forever using finitely many space. However, there is a standard trick to effectively detect such looping in bounded space. So the naive notion can be patched so as to satisfy the axioms. With the definitions of Blum's complexity measures, complexity classes can be generalized as follows:

Definition 91 *Let $\langle \psi_i \rangle_{i \in \mathbf{N}}$ be any acceptable programming system and $\langle \Psi_i \rangle_{i \in \mathbf{N}}$ be a Blum complexity measure associated with $\langle \psi_i \rangle_{i \in \mathbf{N}}$. Let $t : \mathbf{N} \to \mathbf{N}$ be recursive. We say that t **determines** the complexity class*

$$\mathcal{C}(t) = \left\{ f \in \mathcal{R} \,\middle|\, \exists e \left[\psi_e = f \ \text{and} \ \overset{\infty}{\forall} x \left(\Psi_e(x) \le t(|x|) \right) \right] \right\}.$$

■

Thus, the definition above extends Hartmanis and Stearns's original intuitive concept of complexity classes as in (14) on page 65 to any complexity measure that satisfies Blum's axioms.

Recall from Section 1.4.1 that our standard complexity measure, $\langle \Phi \rangle_{i \in \mathbf{N}}$, is the number of steps a TM performs. Clearly, such $\langle \Phi_i \rangle_{i \in \mathbf{N}}$ is a Blum complexity measure associated with our acceptable programming system $\langle\varphi_i\rangle_{i\in\mathbf{N}}$. For simplicity, we use "complexity measure" to mean "Blum complexity measure." If our default complexity measure is taken, then, for any $t \in \mathcal{R}$, $\mathcal{C}(t)$ is denoted as $DTIME(t)$.

The two axioms capture key features of our intuitive notion of dynamic complexity measures. On the other hand, Blum's approach has isolated the complexity theory from specific machine models and dynamic computational resources in type-1 computations. The following theorem states that any two complexity measures are equivalent in a sense that they are recursively related.

Theorem 92 (Recursive Relatedness of Complexity Measures [Blu67]) *Suppose that $\langle\Psi_i\rangle_{i\in\mathbf{N}}$ and $\langle\Gamma_i\rangle_{i\in\mathbf{N}}$ are two complexity measures associated with the acceptable programming system, $\langle\psi_i\rangle_{i\in\mathbf{N}}$. Then, there is a recursive $g : \mathbf{N}\times\mathbf{N}\to\mathbf{N}$ such that, for every $i, x \in \mathbf{N}$, we have*

$$\Psi_i(x) \ \leq^* g(x, \Gamma_i(x)) \quad \textit{and} \quad \Gamma_i(x) \ \leq^* g(x, \Psi_i(x)). \qquad \blacksquare$$

It follows from this theorem that, for any result we prove about a particular complexity measure, there is a "recursively related" result about each other complexity measure. Thus, if our result is invariant[1] with respect to recursive relatedness, then showing it holds for one measure implies that it holds for all measures.[2] We will use our standard acceptable programming system $\langle\varphi_i\rangle_{i\in\mathbf{N}}$ and complexity measure $\langle\Phi_i\rangle_{i\in\mathbf{N}}$ associated to it for the time being.

In the remaining of this chapter we review the most important theorems that follow from the two axioms. We shall prove their type-2 analogies in the next chapter.

[1]For *recursively invariant*, see [Rog92], Chapter 4.

[2]Also note that any two acceptable programming systems are recursively isomorphic, it follows that the two complexity measures in Theorem 92 are not necessarily associated with the same acceptable programming system. See [Odi99], page 24, for details. For more details about the isomorphisms between two acceptable programming systems, see [RC94], page 27, and [Odi89], page 219.

5.2 The Union Theorem

One of the most familiar complexity classes is *PTIME* which is defined as

$$PTIME \;=\; \left\{ f \;\middle|\; f \in DTIME(p), \text{ where } p \text{ is a polynomial} \right\}.$$

Or, equivalently,

$$PTIME \;=\; \bigcup_{k \in \mathbf{N}} DTIME(g_k), \tag{35}$$

where $g_k = \lambda x.[k \cdot (x+1)^k]$. Clearly, the union set in (35) gives us a more intuitive idea about what *PTIME* is. However, it is not at all clear that *PTIME* is indeed a complexity class under the official definition given by Definition 91. In other words, we want to see whether or not we can construct a recursive function that *determines* exactly the same complexity class, *PTIME.* The following theorem known as the *Union Theorem* (first proved by McCreight and Meyer [MM69]) shows that the official notion of complexity classes is indeed broad enough to encompass classes such as *PTIME.*

Theorem 93 (The Union Theorem [MM69]) *Given any sequence of recursive functions* $f_0, f_1, f_2, \ldots$ *such that,*

(i) $\lambda i, x. f_i(x)$ *is recursive, and*

(ii) for all $i, x \in \mathbf{N}$, $f_i(x) \le f_{i+1}(x)$,

there is a recursive function g *such that* $\mathcal{C}(g) = \bigcup_{i \in \mathbf{N}} \mathcal{C}(f_i)$. ■

Clearly, in (35), the g_k's satisfy the condition required in the Union Theorem. Thus, *PTIME* is a complexity class. Some other well-known examples are: the class of exponentially computable functions, the class of elementary recursive functions, the class of primitive recursive functions, and so on. Note that since there is no such uniformly effective enumeration that can cover the entire class of recursive functions, we cannot use the Union Theorem to claim the class of recursive functions a complexity class. In fact, Blum [Blu67] proves that given any recursive t, there always exists a recursive function g such that, $g \notin DTIME(t)$.

5.3 The Compression, Gap, and Honesty Theorems

We want to have a uniform method, by which, given any recursive t, we can construct a larger t' so that $\mathcal{C}(t) \subset \mathcal{C}(t')$. In doing so, there is a surprising subtle problem to face. This section reviews some basic positive and negative results related to this problem.

5.3.1 Measured Sets and The Compression Theorem

Definition 94 (Measured Sets [Blu67]) *A set of partial recursive functions, $\mathcal{M}$, is **measured** if and only if $\mathcal{M} = \{\varphi_{g(i)} | i \in \mathbf{N}\}$, provided*

1. *g is recursive, and*

2. *$\lambda i, x, m.[\varphi_{g(i)}(x) = m]$ is a recursive predicate.*

*We call such set $\mathcal{M}$ a **measured set** determined by g.* ■

For example, if $\langle \psi_i \rangle_{i \in \mathbf{N}}$ is a programming system for the class of polynomials, then $\{\psi_i |\ i \in \mathbf{N}\}$ is a measure set. It is also clear that neither $\mathcal{R}$ nor $\mathcal{PR}$ is a measured set; $\mathcal{R}$ does not satisfy the first property and $\mathcal{PR}$ does not satisfy the second property.

The following theorem is known as the compression theorem. It asserts that, for any measured set, there is a uniform procedure by which, given any recursive function t in the measured set, we can obtain a recursive function that determines a strictly bigger complexity class than t does.

Theorem 95 (The Compression Theorem [Blu67]) *Suppose that $g \in \mathcal{R}$ and g determined a measured set, $\{\varphi_{g(i)} | i \in \mathbf{N}\}$. Then, there is a recursive function $k : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ such that, for all $i \in \mathbf{N}$, if $\varphi_{g(i)}$ is recursive, then we have*

$$\mathcal{C}(\varphi_{g(i)}) \subset \mathcal{C}(\lambda x \cdot k(\varphi_i(x), x)).$$

■

The Compression Theorem can be restated as:

Corollary 96 *Let $g \in \mathcal{R}$ determine a measured set, $\{\varphi_{g(i)} | i \in \mathbf{N}\}$. There is a recursive function $k : \mathbf{N} \to \mathbf{N}$ such that, for all $i \in \mathbf{N}$, if $\varphi_{g(i)}$ is recursive, then $\mathcal{C}(\varphi_{g(i)}) \subset \mathcal{C}(\varphi_{k(i)})$.* ■

The name of the theorem refers to the fact that, there is always a computable function with complexity compressed between $\varphi_{g(i)}$ and $\varphi_{k(i)}$.

5.3.2 The Gap Theorem

Why should we restrict the resource bound functions from a measured set? Intuitively, we can take any recursive function to serve as a resource bound function and uniformly increase the function to obtain a strictly bigger complexity class. However, the Gap Theorem (due to Borodin [Bor69]) shows that this intuition is wrong. The theorem asserts that, given any arbitrarily large recursive function r, there always exists an increasing recursive function t such that, there is no computable function with the complexity falling in the gap between t and $r \circ t$. This is strongly opposite to the compression theorem.

Theorem 97 (The Gap Theorem [Bor69, Bor72]) *For every recursive function $r : \mathbf{N} \to \mathbf{N}$, there is an increasing recursive function $t : \mathbf{N} \to \mathbf{N}$ such that, $\mathcal{C}(t) = \mathcal{C}(r \circ t)$.* ■

Recall that a functional $\Theta : (\mathbf{N} \to \mathbf{N}) \rightharpoonup (\mathbf{N} \to \mathbf{N})$ restricted to $\mathcal{R}$ is said to be an *effective operator* iff Θ is determined by some recursive and extensional $f : \mathbf{N} \to \mathbf{N}$, i.e., $f \in \mathcal{R}$ and, for all $i, j \in \mathbf{N}$, if $\varphi_i = \varphi_j$ then $\varphi_{f(i)} = \varphi_{f(j)}$.[3] The gap theorem was extended to the Operator Gap Theorem.

Theorem 98 (The Operator Gap Theorem [Con72, You73]) *Given any total effective operator $\Theta : (\mathbf{N} \to \mathbf{N}) \to (\mathbf{N} \to \mathbf{N})$, we can effectively find an arbitrarily large recursive function $t : \mathbf{N} \to \mathbf{N}$ such that $\mathcal{C}(t) = \mathcal{C}(\Theta(t))$.* ■

Note that, the recursive function r and the effective operator Θ in Theorems 97 and 98, respectively, can be arbitrarily large. Thus, the two theorems assert that there are infinitely many arbitrarily large empty gaps between complexity classes!

5.3.3 The Honesty Theorem

The Operator Gap Theorem asserts that we cannot have a uniform procedure to construct a recursive function t' from any given recursive function t such that, the complexity class determined by t' is strictly bigger than the one determined by t. Thus, in the Compression Theorem something like the measured set hypothesis is necessary. But, do we lose any complexity class if we restrict resource functions to a measured set in order to remove the gap phenomenon? McCreight and Meyer

[3]See [Rog92], 11.5, page 192, for more details about effective operators.

[MM69] prove by a priority argument that to restrict the resource bound functions in a measured set does not lose any complexity class, provided that we construct the measured set properly. In other words, there is a measured set $\mathcal{M}$ such that, for every recursive function f, there is a recursive function $g \in \mathcal{M}$ such that $\mathcal{C}(f) = \mathcal{C}(g)$. The theorem is called honesty theorem because the measured set and the set of honest functions (discussed below) are intimately related.

Theorem 99 (The Honesty Theorem [MM69]**)** *There exists a recursive function h that determines a measured set, $\{\varphi_{h(i)} |\ i \in \mathbf{N}\}$, such that, for all recursive function f, if $f = \varphi_i$, then $\mathcal{C}(f) = \mathcal{C}(\varphi_{h(i)})$.* ■

A function f is said to be *g-honest* if the function can be computed by some φ_i such that, $\Phi_i \leq^* g \circ f$. That is, f is honest in the sense that, for all but finitely many x, the complexity to compute f on x honestly reflects the value, $f(x)$. McCreight and Meyer [MM69] also prove that every measured set is embedded in a set of g-honest functions, for some recursive function g. In other words, the notion of measured sets is embedded in the notion of honest sets.

Constructability is a preferred property for resource bound functions, which in fact is implied in the notion of honesty. For example, a *time-constructable* function t is a recursive function such that, there is some Turing machine with index i that computes t and, for all $x \in \mathbf{N}$, $\Phi_i(x) = |t(x)|$, where Φ is the time complexity measure. Moreover, if t is a time-constructable function, then $t \in \mathcal{C}(t)$. It is clear that the set of time-constructable functions is honest.

By combining the Honesty Theorem and the Compression Theorem, we have an effective way to obtain a strictly increasing hierarchy of complexity classes from any index of recursive function. Let h be the function stated in the Honesty Theorem. Thus, $\{\varphi_{h(i)} | i \in \mathbf{N}\}$ is a measured set. Let k_h be the function stated in Corollary 96 for the measured set determined by h. Thus, if φ_i is a recursive function, then we have the following strictly increasing complexity classes.

$$\mathcal{C}(\varphi_i) \subset \mathcal{C}(\varphi_{k_h(i)}) \subset \mathcal{C}(\varphi_{k_h^2(i)}) \subset \mathcal{C}(\varphi_{k_h^3(i)}) \subset \cdots\cdots .$$

Constable [Con72][4] points out that no recursive function h as in the honesty theorem can be extensional, i.e., for any such h, there always exist i and j such that, $\varphi_i = \varphi_j$ and $\varphi_{h(i)} \neq \varphi_{h(j)}$. In other words, there is no total effective operator to perform the honesty procedure. Thus, the construction above does not contradict the Operator Gap Theorem, because the function h is not extensional, so neither is k_h, and hence k_h does not determine a total effective operator.

5.4 The Speed-up Theorem

Speed-up phenomena have been extensively studied by mathematicians for more than a half century. A Speed-up Theorem in logical form was first remarked by Gödel [Göd36].[5] For a computable function, a best program to compute it is the program that computes the function with a most efficient way of using the resource. Blum [Blu67] re-discovered a speed-up phenomenon related to this and showed that a best program does not always exist. The Speed-up Theorem asserts that for any acceptable programming system associated with any Blum's complexity measure, there are recursive functions f such that, for any φ-program for f, there always exists another φ-program for f that runs much faster than the old program. More precisely:

[4]Also, see [Van75] for more details on this issue.

[5]Gödel's original article was translated and collected in [Dav65], pages 82-83. More discussion about the relation between the computational Speed-up phenomena and Gödel's speedup results in logic can be found in [Van75].

Theorem 100 (The Speed-up Theorem [Blu67]) *For any recursive function* $r : \mathbf{N} \to \mathbf{N}$*, there exists a recursive function* f *such that*

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

■

We call the function f of the theorem *r-speedable*. Many variations of the Speed-up Theorem have proved since Blum. Meyer and Fischer [MF72] extend the recursive speed-up factors to effective operators.

Theorem 101 (The Operator Speed-up Theorem [MF72]) *For any total effective operator* Θ*, there is a recursive function* f *that can be uniformly constructed such that*

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

■

Since the Speed-up Theorem has nothing to do with the complexity classes and since that type-1 computation is just a special case of type-2 computation, we will see in Chapter 7 that the phenomenon is well preserved in the type-2 computation. We sketch a proof of the original Speed-up Theorem in Section 7.1.2 in order to show that the proof can be translated into our type-2 programing system $\langle \widehat{\varphi}_i \rangle_{i \in \mathbf{N}}$, so that the type-2 analogous theorem holds but not in any particularly interesting.

Chapter 6

Type-2 Complexity Theory

The previous chapter outlined some central themes of the computational complexity theory through several important and classic theorems in this field. When we consider extending the theory into type-2, these theorems shall be translated into their type-2 analogs and be examined carefully to see whether they are still *valid* and *meaningful* in type-2 computations. We avoid a type-2 analog of Blum's axioms in defining type-2 complexity classes. Instead, we take the specific notions of type-2 complexity classes defined in Chapter 4 as our starting point in developing type-2 complexity theory in the next two chapters. The two different cost models for OTMs lead to different results in many places. We summarize our main topics of this chapter as follows:

Union Theorems: We prove that in general the union of two complexity classes is not necessarily a complexity class. Some reasonably strong hypotheses are needed for a union theorem to hold in type-2 under the answer-length-cost model. We define a type-2 analog of the big-O notation, $\mathbf{O}(\beta)$. With some conditions on β, our Type-2 Union Theorem yields that $\mathbf{O}(\beta)$ is a type-2 complexity class. It turns out that $\mathbf{BFF}_2$ fails being a type-2 complexity class. We also point out that the unit-cost-model analog of our Type-2 Union Theorem fails to hold under the same hypotheses, and it seems that extremely strong hypotheses are necessary for any form of the union theorem to hold in this setting. Such strong hypotheses make the union theorem under the unit-cost model less interesting. This adds to the evidence against the unit-cost model being a reasonable cost model for type-2 computation.

Gap Theorems: We first restrict the *gap factor* to type-1 recursive functions. As with the type-2 union theorems, the two OTM cost models have different gap phenomena. Under the answer-length-cost model, the type-2 gap phenomenon is simply inherited from type-1. On the other hand, the type-2 gap phenomenon disappears under the unit-cost model. Moreover, if we consider type-2 gap factors, then a result different from the original Operator Gap Theorem is obtained. We discussion this in the section on Compress Theorems.

Compression Theorems: Intuitively, we can translate the original Operator Gap Theorem to a type-2 gap theorem in which the gap factor is a computable type-2 functional. However, we are interested in the effective operators of type $\mathbf{T_2TB} \to \mathbf{T_2TB}$, and this restriction fails a direct translation from the original Operator Gap Theorem in which the operator concerned is a more general type $(\mathbf{N} \to \mathbf{N}) \to (\mathbf{N} \to \mathbf{N})$ (see Theorem 98 on page 124). In fact, from any $\beta \in \mathbf{T_2TB}$, we can uniformly construct another $\beta' \in \mathbf{T_2TB}$ such that, β' determines a strictly bigger type-2 complexity class than β does. This is indeed an operation analog to the one in the original Compression Theorem. In other words, the type-2 gap phenomenon disappears if the gap factor is an effective operator $\Theta : \mathbf{T_2TB} \to \mathbf{T_2TB}$. We call the type-2 analog of the Compression Theorem *Inflation Theorem.*

The Honesty Theorem: Since our type-2 time bounds are indeed type-1 functions, the main questions about the honesty of our type-2 time bounds are answered by the original study in type-1 complexity theory. We thus do not give special treatment of honesty. In fact, the proof of the Honesty Theorem in our $\widehat{\varphi}$-programming system can be obtained from a directly lifting of the original proof. However, we point out that the Honesty Theorem is no longer interested as one of its important application is not needed without the gap phenomenon in type-2 computation.

Notations and Conventions

Recall that, for each $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{\Phi}_e(f, x)$ denotes the number of steps OTM $\widehat{M}_e$ performs on (f, x), provided that the type-0 input x and every oracle answer have to be scanned at least once. We call this model the answer-length-cost model, and we take it as the default cost model for our OTMs. We use OTM_U to abbreviate "OTM under the unit-cost model". An OTM_U needs to scan its type-0 input at least once, but it scans the oracle answer only on demand. Similarly, we denote the number of steps that OTM_U $\widehat{M}^u_e$ performs on (f, x) by $\widehat{\Phi}^u_e(f, x)$. As in Chapter 2, when the relative topology is clear from the context, we use the following two notations for simplicity:

- $\overset{\infty}{\forall}_{\mathbf{2}}\ (f, x) \ \triangleq$ "for all $(f, x) \in \mathcal{T} \times \mathbf{N}$ except a compact subset of $\mathcal{T} \times \mathbf{N}$ such that"
- $\overset{\infty}{\exists}_{\mathbf{2}}\ (f, x) \ \triangleq$ "there exists a noncompact subset of $\mathcal{T} \times \mathbf{N}$ such that for all (f, x) in the set"

Recall that, for each $f \in \mathcal{T}$, $f_{[n]}$ denotes the initial n-element segment of f, i.e.,

$$f_{[n]} = \left\{ \langle x, f(x) \rangle \;\middle|\; x \in \mathbf{N} \text{ and } x < n \right\}.$$

Cook and Kapron [CK89, Kap91, KC96] define the length of an $f \in \mathcal{T}$ as

$$|f| = \lambda a.\mathsf{max}\left(\left\{ |f(x)| \;\middle|\; |x| \leq a \right\} \right).$$

Recall from Chapter 3 that, for every $\beta \in \mathbf{T_2TB}$, $\beta(\sigma, a)\downarrow$ means that β is *locked* on (σ, a). Definition 60 defines the exception set, $E_{e,\beta}$, according to the β-clocked simulation, $\widehat{\varphi}_{e,\beta}$, which is based on the clocking scheme shown in Figure 3 on page 78 and the base machine, $\widehat{M}_e$, is an $\mathrm{OTM_A}$. We use the similar notations for $\widehat{\varphi}^u_{e,\beta}$ and $E^u_{e,\beta}$ in the same manner except that the base machine, $\widehat{M}^u_e$, in the β-clocked simulation is an $\mathrm{OTM_U}$. For any $\beta \in \mathbf{T_2TB}$, the type-2 complexity class, $\mathbf{C}(\beta)$, is defined by

$$\mathbf{C}(\beta) = \left\{ \widehat{\varphi}_e : \mathcal{T} \times \mathbf{N} \to \mathbf{N} \;\middle|\; e \in \mathbf{N} \text{ and } E_{e,\beta} \text{ is compact in } \mathbb{T}(\widehat{\varphi}_e) \right\},$$

where $\mathbb{T}(\widehat{\varphi}_e)$ is the topology determined by $\widehat{\varphi}_e$ (see Definition 30 and the discussion on page 62.) For convenience,

Definition 102 *Let $\beta \in \mathbf{T_2TB}$, and $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$. We say that the computational complexity of F is* ***bounded*** *by β if and only if $F \in \mathbf{C}(\beta)$.*

For $\sigma \in \mathcal{F}$, $\sigma^{\sim 0}$ denotes the zero extension of σ.

Definition 103 *For each $e \in \mathbf{N}$, define $\widetilde{\Phi}_e : \mathcal{F} \times \mathbf{N} \rightharpoonup \mathbf{N}$ by*

$$\widetilde{\Phi}_e(\sigma, x) = \begin{cases} \widehat{\Phi}_e(\sigma^{\sim 0}, x) & \textit{if all the queries made in the computation of } \widehat{M}_e \textit{ on} \\ & (\sigma^{\sim 0}, x) \textit{ are in } \mathsf{dom}(\sigma); \\ & \\ c & \textit{otherwise,} \quad \textit{where } c \textit{ is the cost of the computation of} \\ & \widehat{M}_e \textit{ on } (\sigma^{\sim 0}, x) \textit{ up to the completion of the first query} \\ & \textit{outside } \mathsf{dom}(\sigma). \end{cases}$$

■

Clearly, if $\widehat{\varphi}_e(\sigma^{\sim 0}, x) \uparrow$ and its computation does not query beyond $\mathsf{dom}(\sigma)$, then, for every $\tau \in \mathcal{F}$ with $\sigma \subseteq \tau$, $\widetilde{\Phi}_e(\tau, x) \uparrow$. Also, note that if $\widehat{\varphi}_{e,\beta}(f, x) \Downarrow$, then for every $\sigma \subset f$, $\widetilde{\Phi}_e(\sigma, x) \leq \beta(\sigma, x)$. Similarly, we use $\widetilde{\Phi}_e^u$ when an $\mathrm{OTM_U}$ is considered.

6.1 Union vs. Non-union Theorems

We have seen that the type-1 Union Theorem is a useful tool to show that the notion of complexity class is broad enough to encompass natural computational classes such as *PTIME*. In type-2 the most widely studied "complexity class" is $\mathbf{BFF}_2$, and Cook and Kapron [CK89, Kap91, KC96] give a very intuitive analog between *PTIME* and $\mathbf{BFF}_2$. Thus, it seems inevitable that we should find out whether $\mathbf{BFF}_2$ is a type-2 complexity class under our setting.

Before we consider the situation in details, we first give the following terminologies that will help in our discussion. Given $\beta_1, \beta_2 \in \mathbf{T}_2\mathbf{TB}$, $\beta_1 \leq \beta_2$ means that, for every $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, $\beta_1(\sigma, x) \leq \beta_2(\sigma, x)$. Recall Definition 56 and Theorem 59. For each $\beta \in \mathbf{T}_2\mathbf{TB}$, we say that β is ***useful*** if and only if, for every (σ, x) such that, (σ, x) is a minimal locking fragment of β, we have

$$\forall \tau \subseteq \sigma[\beta(\tau, x) \geq \|\tau\| + |x| + 1].$$

Definition 104 *Let $\langle \beta_i \rangle$ denote a sequence of type-2 time bounds $\beta_0, \beta_1, \beta_2, \ldots$.*

1. *We say that $\langle \beta_i \rangle$ is **uniform** if and only if $\lambda i, \sigma, x.\beta_i(\sigma, x)$ is recursive.*
2. *We say that $\langle \beta_i \rangle$ is **ascending** if and only if, for all $i \in \mathbf{N}$, $\beta_i \leq \beta_{i+1}$.*
3. *We say that $\langle \beta_i \rangle$ is **useful** if and only if, for all $i \in \mathbf{N}$, β_i is useful.*
4. *We say that $\langle \beta_i \rangle$ is **convergent** if and only if, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, there is a $\sigma \subset f$ such that, $\beta_i(\sigma, x) \downarrow$ for every $i \in \mathbf{N}$.*

5. *We say that $\langle \beta_i \rangle$ is **uniformly convergent** if and only if, for every $n \in \mathbf{N}$ and $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, if $\beta_n(\sigma, x) \downarrow$, then for all $i \in \mathbf{N}$, $\beta_i(\sigma, x) \downarrow$.*

6. *We say that $\langle \beta_i \rangle$ is **strongly convergent** if and only if $\langle \beta_i \rangle$ is uniformly convergent and there is a recursive locking detector for β_0.* ■

Example 105 *For every $i \in \mathbf{N}$ and $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, define*

$$\alpha_i(\sigma, x) = \begin{cases} \sigma(x) + |x|^{i+1} + 1 & \text{if } x \in \mathsf{dom}(\sigma); \\ |x|^{i+1} + 1 & \text{otherwise.} \end{cases}$$

$$\beta_i(\sigma, x) = \begin{cases} \sigma(x+i) + |x|^{i+1} + 1 & \text{if } (x+i) \in \mathsf{dom}(\sigma); \\ |x|^{i+1} + 1 & \text{otherwise.} \end{cases}$$

One can see that $\langle \alpha_i \rangle$ and $\langle \beta_i \rangle$ above are two sequences of type-2 time bounds. Clearly, $\langle \alpha_i \rangle$ is *uniform*, *ascending*, and *strongly convergent*, while $\langle \beta_i \rangle$ is *uniform*, but neither *ascending* nor *convergent*. Moreover, all type-2 time bounds in $\langle \alpha_i \rangle$ and $\langle \beta_i \rangle$ are useful. ■

Let $\langle \beta_i \rangle$ be strongly convergent. Suppose that ℓ is a locking detector for β_0. By definition, $\langle \beta_i \rangle$ is uniformly convergent. Thus, we can use ℓ to detect the convergence of the entire sequence. That is,

$$[\ell(\sigma, x) = 1] \implies \forall i \in \mathbf{N}[\beta_i(\sigma, x) \downarrow],$$

and, for all $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\lim_{\sigma \to f} \ell(\sigma, x) = 1$.

Lemma 106 *Given uniform and ascending $\langle \beta_i \rangle$, if there exists a total continuous functional $H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that, for every $i \in \mathbf{N}$, $F_{\beta_i} \leq H$, then $\langle \beta_i \rangle$ is convergent.*

Proof: Suppose by contradiction that $\langle\beta_i\rangle$ is not convergent. Thus, there is a $(f, x) \in \mathcal{T} \times \mathbf{N}$ such that, there is no (σ, x) with $\sigma \subset f$ that makes every β_i to converge on it. That is, for each $i \in \mathbf{N}$, if $\sigma_i \subset f$ and (σ_i, x) is a minimal locking fragment of β_i, then there exists $j > i$ such that, (σ_i, x) is not a locking fragment of β_j. Let σ_i, i and j be such. Since $\langle\beta_i\rangle$ is ascending, we have

$$\beta_i(\sigma_i, x) = F_{\beta_i}(f, x) \leq \beta_j(\sigma_i, x) < F_{\beta_j}(f, x).$$

By assumption, $\lim\limits_{i\to\infty} F_{\beta_i}(f, x) \leq H(f, x)$, and hence $\lim\limits_{i\to\infty} F_{\beta_i}(f, x)$ is not bounded. But, since H is a total continuous functional, $H(f, x) \downarrow = y$ for some finite number y. This leads a contradiction, and hence $\langle\beta_i\rangle$ cannot be convergent. $\square$

Given any $\beta \in \mathbf{T}_2\mathbf{TB}$, define $\langle\beta_i\rangle$ as, for each $i \in \mathbf{N}$, let $\beta_i = i\beta$. It is clear that such $\langle\beta_i\rangle$ is a counterexample of the inverse of Lemma 106.

Recall from Definition 35 on page 60 that, for any two continuous $F, G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, if the set $\{(f, x) \mid F(f, x) > G(f, x)\}$ is compact in $\mathbb{T}(F, G)$, we say that F is almost everywhere less than G, denoted as $F \leq_{\mathbf{2}}^{*} G$. If we relax $F_{\beta_i} \leq H$ in Lemma 106 to $F_{\beta_i} \leq_{\mathbf{2}}^{*} H$, then we have the following lemma which is stronger than the inverse of Lemma 106 in the sense that we do not require $\langle\beta_i\rangle$ to be convergent.

Lemma 107 *For any uniform and ascending $\langle\beta_i\rangle$, there is a total continuous functional $H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that, for every $i \in \mathbf{N}$, $F_{\beta_i} \leq_{\mathbf{2}}^{*} H$.*

Proof: Define $H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ by

$$H(f, x) = F_{\beta_{(x+\|\tau\|)}}(f, x),$$

where $\tau \subset f$ and (τ, x) is a minimal locking fragment of β_x. It is clear that such H is total and continuous.

Fix any $i \in \mathbf{N}$. We shall prove that $F_{\beta_i} \leq_{\mathbf{2}}^{*} H$. Fix $(f, x) \in \mathcal{T} \times \mathbf{N}$ and let $\tau \subset f$ be such that (τ, x) is a minimal locking fragment of β_x. We have two cases: $i \leq x$ and $x < i$.

Case 1: $i \leq x$. Since $\langle \beta_i \rangle$ is ascending, it is clear that,

$$F_{\beta_i}(f, x) \leq F_{\beta_x}(f, x) \leq F_{\beta_{x+\|\tau\|}}(f, x) = H(f, x).$$

Case 2: $x < i$. Suppose that $H(f, x) < F_{\beta_i}(f, x)$, i.e.,

$$H(f, x) = F_{\beta_{x+\|\tau\|}}(f, x) < F_{\beta_i}(f, x).$$

There are finitely many different (τ, x)'s such that, (τ, x) is a locking fragment of β_i and $\|\tau\| \leq i - x$. Therefore, $\{(f, x) | F_{\beta_i}(f, x) > H(f, x)\}$ is compact in $\mathbb{T}(F_{\beta_i}, H)$, and hence $F_{\beta_i} \leq_{\mathbf{2}}^{*} H$. □

Note that the functional H in Lemma 107 is not necessarily computable unless we can effectively determine when does each β_i converge. If we can, then H is computable since $\langle \beta_i \rangle$ is uniform and, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, the minimal locking fragment (τ, x) and $F_{\beta_{(x+\|\tau\|)}}(f, x)$ can be effectively obtained.

According to Theorem 67, if $\langle \beta_i \rangle$ is ascending, then, for every $i \in \mathbf{N}$, $\mathbf{C}(\beta_i) \subseteq \mathbf{C}(\beta_{i+1})$. Clearly, if $\langle \beta_i \rangle$ is strongly convergent with a locking detector ℓ, then each β_i is a strong type-2 time bound because each β_i can share the same locking detector ℓ. Strong convergence is a strong restriction on $\langle \beta_i \rangle$ but it turns out to be one of the necessary hypotheses in our type-2 analog of the Union Theorem. We now define the union class determined by $\langle \beta_i \rangle$ in the following definition.

Definition 108 *Given any $\langle \beta_i \rangle$, let $\mathbf{C}(\langle \beta_i \rangle)$ denote the union class $\bigcup_{i \in \mathbf{N}} \mathbf{C}(\beta_i)$.* ■

We first show that this definition can be used to describe $\mathbf{BFF}_2$ by a $\langle\beta_i\rangle$. Instead of proving some nontrivial properties about the relation between the second-order polynomial and $\mathbf{BFF}_2$, we simply use some of Cook and Kapron's theorems [CK89, Kap91, KC96]. Also, some proofs with more details that are closer to our setup can be found in [IKR01a].

Theorem 109 *There is a uniform and ascending $\langle\beta_i\rangle$ such that, $\mathbf{C}(\langle\beta_i\rangle) = \mathbf{BFF}_2$.*

Proof: For each $k \in \mathbf{N}$, let p_k be the two-variable polynomial defined by

$$p_k(m,n) = k(m+n+1)^k.$$

Here we follow the definition of $q^{d,k}$ in [IKR01a]. For every $d, k \in \mathbf{N}$, we recursively define the second-order polynomial $q^{d,k}$ the following.

For every $g : \mathbf{N} \to \mathbf{N}$ and $y \in \mathbf{N}$,

$$\begin{aligned} q^{0,k} =& p_k(0,y). \\ q^{d+1,k} =& p_k(g(q^{d,k}(g,y)),y). \end{aligned}$$

By a straightforward argument, we have that, for every second-order polynomial $\boldsymbol{q}$ over $f : \mathbf{N} \to \mathbf{N}$ and $x \in \mathbf{N}$, if d is the depth of $\boldsymbol{q}$, then there is a $k \in \mathbf{N}$ such that, for every f and x,

$$\boldsymbol{q}(|f|,|x|) \leq q^{d,k}(|f|,|x|).$$

For each $i \in \mathbf{N}$, define $\beta_i : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ by

$$\beta_i(\sigma, x) = \mathsf{max}(|x|+1, q^{i,i}(|\sigma^{\sim 0}|,|x|)).$$

Since, for each $i \in \mathbf{N}$ and $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, $q^{i,i}(|\sigma^{\sim 0}|,|x|)$ is computable, β_i is computable and hence $\langle\beta_i\rangle$ is uniform. It is also clear that β_i is nontrivial, strong $\mathcal{F}$-monotone, and convergent. Moreover, since if $i \leq j$, then $q^{i,i} \leq q^{j,j}$, it follows that $\langle\beta_i\rangle$ is ascending.

If (σ, x) is a locking fragment of β_i, then, for all $f \supset \sigma$, we have

$$F_{\beta_i}(f,x) = \mathsf{max}(|x|+1, q^{i,i}(|f|,|x|)).$$

Thus, F_{β_i} is a second-order polynomial of depth i. Moreover, for every second-order polynomial $\boldsymbol{q}$ such that, for every $(f,x) \in \mathcal{T} \times \mathbf{N}$, $\boldsymbol{q}(|f|,|x|) \leq q^{d,k}(|f|,|x|)$, we have $\boldsymbol{q} \leq q^{d,k} \leq F_{\beta_i}$, where $i = \mathsf{max}(d,k)$. By the Lemma for the *continuity of bounds* in [IKR01a], if the functional F can be computed in time bounded by F_{β_i}, then $F \in \mathbf{C}(\beta_i)$. Therefore, F is bounded by some β_i if and only if there is an OMT $\widehat{M_e}$ that computes F and its runtime is bounded by a second-order polynomial. □

Corollary 110 *There is a programming system for* $\mathbf{BFF}_2$.

Sketch of Proof: Let $\langle\beta_i\rangle$ be the sequence of type-2 time bounds in Theorem 109. We can simply use our locking scheme to clock every OTM $\widehat{M_e}$ with β_e. □

Lemma 111 *For any* $\langle\beta_i\rangle$ *such that* $\mathbf{C}(\langle\beta_i\rangle) = \mathbf{BFF}_2$, $\langle\beta_i\rangle$ *is not convergent.*

Proof: Fix $\langle\beta_i\rangle$ such that $\mathbf{C}(\langle\beta_i\rangle) = \mathbf{BFF}_2$. Clearly, the depth of a second-order polynomial can be arbitrarily large. If we fix $i, x \in \mathbf{N}$ and a strictly increasing $f \in \mathcal{T}$ and let (σ, x) with $\sigma \subset f$ be a locking fragment of β_i, then there always exists some second-order polynomials need to refer to some values of f beyond σ. Thus, there must be some β_j that does not converge on (σ, x), and hence $\langle\beta_i\rangle$ cannot be convergent. □

The class *PTIME* is usually understood as the union of complexity classes shown in the right-hand-side of (35) on page 121. By applying the original union theorem, we can formally consider *PTIME* as a complexity class. Similarly, $\mathbf{BFF}_2$, by Cook and Kapron's theorems, can be viewed as a union of complexity classes each of which is determined by a second order polynomial. In our setting, we have $\mathbf{BFF}_2 = \mathbf{C}(\langle\beta_i\rangle)$

for some uniform and ascending $\langle\beta_i\rangle$. However, we will see in Section 6.1.2 that $\mathbf{BFF}_2$ is not a type-2 complexity class according to our official definitions.

Just like type-1 computations, there is no β that determines a complexity class that includes all total computable type-2 functionals. The next theorem shows that, even when a given $\langle\beta_i\rangle$ is *uniform* and *ascending*, the union class, $\mathbf{C}(\langle\beta_i\rangle)$ cannot include all computable type-2 functionals.

Theorem 112 *Given any uniform, ascending $\langle\beta_i\rangle$, there is a computable functional $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, such that $F \notin \mathbf{C}(\langle\beta_i\rangle)$.*

Proof: Define $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ by

$$F(f,x) = 2^{\beta_x(f_{[x]},x)+1}.$$

Clearly, F is computable and total because f is total and every β_i is recursive. Suppose that p is a $\widehat{\varphi}$-program such that, $\widehat{\varphi}_p = F$ and, for each $(f,x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{\varphi}_p(f,x)$ converges at $(f_{[x]},x)$, i.e., $\widehat{\varphi}_p(f,x)$ does not need information about f beyond $f_{[x]}$.

Suppose by contradiction that $\widehat{\varphi}_p \in \mathbf{C}(\beta_i)$ for some $i \in \mathbf{N}$. Thus, the type-0 component of the elements in E_{p,β_i} is bounded, and hence we have

$$\exists x_0 \forall x \geq x_0 \forall f \forall n[\widetilde{\Phi}_p(f_{[n]},x) \leq \beta_i(f_{[n]},x)],$$

which implies that

$$\exists x_0 \forall x > \mathsf{max}(i,x_0) \forall f[\widetilde{\Phi}_p(f_{[x]},x) \leq \beta_i(f_{[x]},x)]. \tag{36}$$

Since $\widehat{\varphi}_p(f,x)$ converges at $(f_{[x]},x)$, it follows that $\widetilde{\Phi}_p(f_{[x]},x) = \widehat{\Phi}_p(f,x)$. From the definition of F, we know that $\beta_x(f_{[x]},x) + 1 \leq \widehat{\Phi}_p(f,x)$. Therefore,

$$\begin{aligned}(36) \;&\Rightarrow\; \exists x_0 \forall x > \mathsf{max}(i,x_0) \forall f[\beta_x(f_{[x]},x) + 1 \leq \beta_i(f_{[x]},x)] \\ &\Rightarrow\; \exists x_0 \forall x > \mathsf{max}(i,x_0) \forall f[\beta_x(f_{[x]},x) < \beta_i(f_{[x]},x)].\end{aligned}$$

```
Program for F
    input f : T; x : N;
    Q ⟵ ∅;

/* The following for-loop assures that, for a fixed f, each index should be
   canceled at most once. */

    for w = 0 to x do
        S_w ⟵ { k | k ≤ w, k ∉ Q, φ̂_{k,β_{f(0)}}(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 */
    return 1 ∸ φ̂_{e,β_{f(0)}}(f, x);
End program
```

Figure 6: A Program for $\widehat{\varphi}_e \notin \mathbf{C}(\langle\beta_i\rangle)$

This contradicts the definition of $\langle\beta_i\rangle$. Therefore, $\widehat{\varphi}_p \notin \mathbf{C}(\beta_i)$ □

A cancellation method is used to prove Theorem 82 asserting that there is an arbitrarily complex total computable functionals which is 0-1 valued. By using the same method, Theorem 82 can be strengthened as follows:

Theorem 113 *Given any uniform and ascending $\langle\beta_i\rangle$, there is a computable functional $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that, $F \notin \mathbf{C}(\langle\beta_i\rangle)$ and F is 0-1 valued.*

Sketch of Proof: Let $\langle\beta_i\rangle$ be uniform and ascending. Consider the type-2 functional F computed by the program shown in Figure 6, which is slightly modified from the algorithm shown in Figure 4 on page 111.

Let $\widehat{\varphi}_e = F$. Suppose by contradiction that $\widehat{\varphi}_e \in \mathbf{C}(\beta_n)$ for some $n \in \mathbf{N}$. Note that if $\widehat{\varphi}_e \in \mathbf{C}(\beta_n)$, then there must exist $x_0 \in \mathbf{N}$ such that,

$$\forall x \geq x_0 \; \forall f \; \widehat{\varphi}_{e,\beta_n}(f, x) \Downarrow .$$

Let x_0 be such a number. Fix $f \in \mathcal{T}$ with $f(0) = n$. Thus, $\forall x \geq x_0\ \widehat{\varphi}_{e,\beta_n}(f, x) \Downarrow$. Then, by an argument similar for that of Theorem 82, it follows that $\widehat{\varphi}_e \neq F$. $\square$

6.1.1 The Weak Type-2 Non-union Theorem

Notation: Given $\beta \in \mathbf{T_2TB}$, $\mathbf{C_A}(\beta)$ denotes the complexity class determined by β under the answer-length-cost model, and $\mathbf{C_U}(\beta)$ the complexity class under the unit-cost model. In contexts where the difference between two models is of no importance, we then simply use $\mathbf{C}(\beta)$.

Just as with the type-1 theory, in general, the union of two arbitrary complexity classes is not always a complexity class. In this sub-section we consider such non-union results. We will see in the next sub-section that a hypothesis is needed in order to obtain a type-2 union theorem.

Theorem 114 (Weak Type-2 Non-union Theorem) *There are two type-2 time bounds, β_1 and β_2 such that,*

$$\forall \alpha \in \mathbf{T_2TB}, \mathbf{C}(\alpha) \neq \mathbf{C}(\beta_1) \cup \mathbf{C}(\beta_2).$$

Proof: Define $F_1, F_2, F_3 : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, respectively, by

$$F_1(f, x) = \begin{cases} 2^{3\cdot|f(x)|} & \text{if } x \text{ is odd;} \\ 2^{|f(x)|} & \text{otherwise,} \end{cases}$$

$$F_2(f, x) = \begin{cases} 2^{3\cdot|f(x)|} & \text{if } x \text{ is even;} \\ 2^{|f(x)|} & \text{otherwise,} \end{cases}$$

$$F_3(f, x) = 2^{2\cdot|f(x)|}.$$

It is clear that we can choose e_1, e_2, and e_3 so that $\widehat{\varphi}_{e_1} = F_1, \widehat{\varphi}_{e_2} = F_2$, $\widehat{\varphi}_{e_3} = F_3$, and the complexity of each program is bounded as follows, for each $(f, x) \in \mathcal{T} \times \mathbf{N}$:

$$\widehat{\Phi}_{e_1}(f,x) \;\leq\; \begin{cases} |x| + 3 \cdot |f(x)| + c & \text{if } x \text{ is odd;} \\ |x| + |f(x)| + c & \text{otherwise,} \end{cases}$$

$$\widehat{\Phi}_{e_2}(f,x) \;\leq\; \begin{cases} |x| + 3 \cdot |f(x)| + c & \text{if } x \text{ is even;} \\ |x| + |f(x)| + c & \text{otherwise,} \end{cases}$$

$$\widehat{\Phi}_{e_3}(f,x) \;\leq\; |x| + 2 \cdot |f(x)| + c,$$

where $c \geq 1$ is some constant. Define $\beta_1, \beta_2 : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ by:

$$\beta_1(\sigma,x) \;=\; \begin{cases} |x| + 3 \cdot |\sigma(x)| + c & \text{if } x \text{ is odd and } x \in \mathsf{dom}(\sigma); \\ |x| + |\sigma(x)| + c & \text{if } x \text{ is even and } x \in \mathsf{dom}(\sigma); \\ |x| + c & \text{otherwise,} \end{cases}$$

$$\beta_2(\sigma,x) \;=\; \begin{cases} |x| + 3 \cdot |\sigma(x)| + c & \text{if } x \text{ is even and } x \in \mathsf{dom}(\sigma); \\ |x| + |\sigma(x)| + c & \text{if } x \text{ is odd and } x \in \mathsf{dom}(\sigma); \\ |x| + c & \text{otherwise.} \end{cases}$$

Clearly, $\beta_1, \beta_2 \in \mathbf{T}_2\mathbf{TB}$. It is also clear that $\widehat{\varphi}_{e_1} \in \mathbf{C}(\beta_1)$ and $\widehat{\varphi}_{e_2} \in \mathbf{C}(\beta_2)$. Since there are infinitely many odd x such that, for some $f \in \mathcal{T}$,

$$F_{\beta_2}(f,x) \;=\; |x| + |f(x)| + c \;<\; \widehat{\Phi}_{e_1}(f,x) \;\leq\; |x| + 3 \cdot |f(x)| + c,$$

we have that $\widehat{\varphi}_{e_1} \notin \mathbf{C}(\beta_2)$. Similarly, $\widehat{\varphi}_{e_2} \notin \mathbf{C}(\beta_1)$, and $\widehat{\varphi}_{e_3} \notin \mathbf{C}(\beta_1) \cup \mathbf{C}(\beta_2)$.

By contradiction, suppose that there is an $\alpha \in \mathbf{T}_2\mathbf{TB}$ such that, $\mathbf{C}(\alpha) = \mathbf{C}(\beta_1) \cup \mathbf{C}(\beta_2)$. That means, $F_1, F_2 \in \mathbf{C}(\alpha)$, and $F_3 \notin \mathbf{C}(\alpha)$. By definition, $E_{e_1,\alpha}$ and $E_{e_2,\alpha}$ are compact in $\mathbb{T}(\widehat{\varphi}_{e_1})$ and $\mathbb{T}(\widehat{\varphi}_{e_2})$, respectively, and $E_{e_3,\alpha}$ is noncompact in $\mathbb{T}(\widehat{\varphi}_{e_3})$. Because F_1, F_2, and F_3 have the same minimal locking fragment on every $(f, x) \in$

$\mathcal{T} \times \mathbf{N}$, we have that $\mathbb{T}(\widehat{\varphi}_{e_1}) = \mathbb{T}(\widehat{\varphi}_{e_2}) = \mathbb{T}(\widehat{\varphi}_{e_3})$. For simplicity, let $\mathbb{T}$ denote $\mathbb{T}(\widehat{\varphi}_{e_1})$, $\mathbb{T}(\widehat{\varphi}_{e_2})$, or $\mathbb{T}(\widehat{\varphi}_{e_3})$, without confusion.

Define $\mathcal{B}_o, \mathcal{B}_e \subseteq \mathcal{F} \times \mathbf{N}$ by:

$$\mathcal{B}_o = \left\{ (\!(\sigma, x)\!) \;\middle|\; x \text{ is odd and } \mathsf{dom}(\sigma) = \{x\} \right\},$$

$$\mathcal{B}_e = \left\{ (\!(\sigma, x)\!) \;\middle|\; x \text{ is even and } \mathsf{dom}(\sigma) = \{x\} \right\}.$$

Let $\mathcal{B} = \mathcal{B}_o \cup \mathcal{B}_e$. Thus, $\mathcal{B}$ is the basis of $\mathbb{T}$ consisting of minimal locking fragments of $\widehat{\varphi}_{e_1}$, $\widehat{\varphi}_{e_2}$, and $\widehat{\varphi}_{e_3}$. Let $\mathcal{O} \subseteq \mathbb{T}$ be an open cover for $E_{e_3,\alpha}$ without any finite subcover. Suppose

$$\mathcal{O} = \{(\!(\sigma_1, x_1)\!), (\!(\sigma_2, x_2)\!), \dots\},$$

where (σ_i, x_i) is a minimal locking fragment of $\widehat{\varphi}_{e_3}$ for every $i \in \mathbf{N}$. Note that, for every $i \in \mathbf{N}$, $\mathsf{dom}(\sigma_i) = \{x_i\}$, and (σ_i, x_i) is also a minimal locking fragment of $\widehat{\varphi}_{e_1}$ and $\widehat{\varphi}_{e_2}$. By the assumption on $\mathcal{O}$, there are infinitely many $(\!(\sigma_i, x_i)\!) \in \mathcal{O}$ such that, the clock α cannot provide enough budget for the computation of $\widehat{\varphi}_{e_3,\alpha}$ on $(\sigma_i^{\sim 0}, x_i)$. Since $\widehat{\Phi}_{e_3}(\sigma_i^{\sim 0}, x_i) \le |x| + 2 \cdot |\sigma_i(x_i)| + c$, we have $\alpha(\sigma_i, x_i) < |x| + 2 \cdot |\sigma_i(x_i)| + c$. In other words,

$$\overset{\infty}{\exists}\, (\!(\sigma, x)\!) \in \mathcal{B}\ \left[\alpha(\sigma, x) < |x| + 2 \cdot |\sigma(x)| + c\right]. \tag{37}$$

Since $E_{e_1,\alpha}$ is compact in $\mathbb{T}$, it follows that, for all but finitely many $(\!(\sigma, x)\!) \in \mathcal{B}$, we have $\widehat{\Phi}_{e_1}(\sigma^{\sim 0}, x) \le \alpha(\sigma, x)$. Thus,

$$\overset{\infty}{\forall}\, (\!(\sigma, x)\!) \in \mathcal{B}_o\ \left[|x| + 3 \cdot |\sigma(x)| + c \le \alpha(\sigma, x)\right]. \tag{38}$$

Similarly, consider $E_{e_2,\alpha}$. We have

$$\overset{\infty}{\forall}\, (\!(\sigma, x)\!) \in \mathcal{B}_e\ \left[|x| + 3 \cdot |\sigma(x)| + c \le \alpha(\sigma, x)\right]. \tag{39}$$

From (38) and (39), we have

$$\overset{\infty}{\forall} \, ((\sigma, x)) \in \mathcal{B} \, \left[|x| + 3 \cdot |\sigma(x)| + c \leq \alpha(\sigma, x) \right]. \tag{40}$$

Clearly, (40) and (37) is a contradiction. Therefore, no such α exists. □

In the next sub-section we will see some stronger versions of the Type-2 Non-union Theorem. Those negative results help us to find a reasonably strong hypotheses for obtaining a type-2 union theorem.

6.1.2 The Non-Union Theorem under The Answer-Length-Cost Model

In the previous sub-section we showed that the union of any two arbitrary complexity classes is not necessarily a complexity class. We next consider stronger versions of this non-union result in the contexts of the answer-length-cost model. These negative results help to justify the rather strong hypotheses of the Type-2 Union Theorem shown in Section 6.1.3.

Theorem 115 (Type-2 Non-Union Theorem) *There is a uniform, ascending, useful, and convergent $\langle \beta_i \rangle$, such that $\mathbf{C_A}(\langle \beta_i \rangle)$ is not a type-2 complexity class.*

Proof: Define $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ by

$$F(f, x) = \begin{cases} 1 & \text{if } f(0) = f(1) = \cdots = f(x) = 0; \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, F is computable and total. Let e be a $\widehat{\varphi}$-program for F such that, on every (f, x), e does not make any unnecessary query outside $\{0, 1, \ldots, x\}$, and

$$\widehat{\Phi}_e(f, x) \leq \|f_{[x+1]}\| + |x| + c,$$

for some constant $c \geq 1$. It is clear that such an e exists. Define $\langle \beta_i \rangle$ by:

$$\beta_i(\sigma, x) = \begin{cases} |x| + c & \text{if } x \geq i; \\ \|\sigma_{[x+2]}\| + |x| + i & \text{otherwise.} \end{cases}$$

Clearly, $\langle \beta_i \rangle$ is uniform, ascending, and useful. For convergence, given $(f, x) \in \mathcal{T} \times \mathbf{N}$, it follows that $f_{[x+2]}$ is a locking fragment for all β_i. Note that every β_i is locking detectable, but $\langle \beta_i \rangle$ is neither strongly nor uniformly convergent. For example, the minimal locking fragment for β_0 is $(\emptyset, x)$, while for β_i with $i > x$ the minimal locking fragment is (σ, x) where $\mathsf{dom}(\sigma) = \{0, 1, \dots, x + 1\}$.

Claim 1: $F \notin \mathbf{C}_{\mathbf{A}}(\langle \beta_i \rangle)$.

Fix any $i \in \mathbf{N}$. For every $(f, x) \in \mathcal{T} \times \mathbf{N}$, if $x \geq i$, then $\widehat{\varphi}_{e,\beta_i}(f, x) \Uparrow$. Thus, for any $i \in \mathbf{N}$, $F \notin \mathbf{C}_{\mathbf{A}}(\beta_i)$, and hence $F \notin \mathbf{C}_{\mathbf{A}}(\langle \beta_i \rangle)$.

Claim 2: For all $\alpha \in \mathbf{T}_2\mathbf{TB}$, $\mathbf{C}_{\mathbf{A}}(\alpha) \neq \mathbf{C}_{\mathbf{A}}(\langle \beta_i \rangle)$.

By contradiction, suppose that $\mathbf{C}_{\mathbf{A}}(\alpha) = \mathbf{C}_{\mathbf{A}}(\langle \beta_i \rangle)$. It follows that $F \notin \mathbf{C}_{\mathbf{A}}(\alpha)$. Fix any $(g, a) \in \mathcal{T} \times \mathbf{N}$ such that, $\widehat{\varphi}_{e,\alpha}(g, a) \Uparrow$. Thus, there must exist $\sigma \subseteq g_{[a+1]}$ such that,

$$\alpha(\sigma, a) < \|\sigma\| + |a| + c.$$

Otherwise, $\widehat{\varphi}_{e,\alpha}(g, a) \Downarrow$. Note that $a + 1 \notin \mathsf{dom}(\sigma)$.

Consider the follow computable functional $F' : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$,

$$F'(f, x) = \begin{cases} 1 & \text{if } x = a, \sigma \subset f, \text{ and } f(a + 1) = 0; \\ 0 & \text{otherwise.} \end{cases}$$

If e' is a $\widehat{\varphi}$-program for F', then, for every $f \in \mathcal{T}$ with $\sigma \subset f$, we have

$$\widehat{\Phi}_{e'}(f, a) > \|\sigma\| + |a| + |f(a + 1)|.$$

Clearly, no $\widehat{\varphi}$-program can compute F' with complexity less than $\|\sigma\| + |a| + |f(a+1)|$ on $(\!(\sigma, a)\!)$. Thus,

$$\forall (f, a) \in (\!(\sigma, a)\!)\ [\widehat{\varphi}_{e',\alpha}(f, a) \Uparrow],$$

and hence $(\!(\sigma, a)\!) \subseteq E_{e',\alpha}$. Since (σ, a) is not a locking fragment of $\widehat{\varphi}_{e'}$, it follows $(\!(\sigma, a)\!)$ is noncompact in $\mathbb{T}(\widehat{\varphi}_{e'})$. By the contrapositive of Lemma 34, we conclude that $E_{e',\alpha}$ is noncompact in $\mathbb{T}(\widehat{\varphi}_{e'})$. Therefore, $F' \notin \mathbf{C_A}(\alpha)$.

On the other hand, we can have a $\widehat{\varphi}$-program e' for F' such that, for every $(f, x) = \mathcal{T} \times \mathbf{N}$, we have

$$\widehat{\Phi}_{e'}(f, x) \leq \begin{cases} |x| + c & \text{if } x \neq a, \\ \|f_{[a+2]}\| + |a| + c & \text{if } x = a. \end{cases}$$

Thus, if we select a sufficiently large $i \geq \mathsf{max}(c, a)$, then $\widehat{\varphi}_{e'} \in \mathbf{C_A}(\beta_i)$ and hence $F' \in \mathbf{C_A}(\langle\beta_i\rangle)$. This contradicts our assumption that $\mathbf{C_A}(\alpha) = \mathbf{C_A}(\langle\beta_i\rangle)$. Therefore, $\mathbf{C_A}(\alpha) \neq \mathbf{C_A}(\langle\beta_i\rangle)$. □

Although a $\langle\beta_i\rangle$ being convergent is already a strong hypothesis, the above theorem shows that convergence plus usefulness are together not sufficient for any type-2 Union Theorem. Thus, we try to further strengthen the hypothesis by including *uniform convergence*. If we drop the *usefulness* in the hypotheses, then we can simply modify the proof of Theorem 115 and have the following negative result.

Corollary 116 *There is a uniform, ascending, and uniformly convergent $\langle\beta_i\rangle$, such that $\mathbf{C_A}(\langle\beta_i\rangle)$ is not a type-2 complexity class.*

Sketch of Proof: We can use the exact arguments for Theorem 115 with the following modification on $\langle\beta_i\rangle$. For every $i, x \in \mathbf{N}$ and $\sigma \in \mathcal{F}$, define

$$\beta_i(\sigma, x) = \begin{cases} \|\sigma_{[x+2]}\| + |x| + i & \text{if } x < i; \\ |x| + c + 1 & \text{if } x \geq i \text{ and } \sigma(0) = \sigma(1) = \cdots = \sigma(x+1) = 0; \\ |x| + c & \text{otherwise.} \end{cases}$$

It is clear that every β_i is to converge at the same fragment. Note that, $\langle\beta_i\rangle$ is no longer useful. □

Thus, the *usefulness* of $\langle\beta_i\rangle$ shall be added to the necessary hypotheses of our Type-2 Union Theorem. However, it is unclear that usefulness together with uniform convergence are sufficient to obtain a type-2 union theorem.

Conjecture 117 *There is a uniform, ascending, useful, and uniformly convergent $\langle\beta_i\rangle$, such that $\mathbf{C_A}(\langle\beta_i\rangle)$ is not a type-2 complexity class.*

It follows from the Union Theorem that the class of type-1 polynomial-time computable functions is a complexity class. We also know from Theorem 109 that $\mathbf{BFF}_2 = \mathbf{C}(\langle\beta_i\rangle)$ for a particular uniform and ascending $\langle\beta_i\rangle$. An obvious question is whether $\mathbf{BFF}_2$ is a type-2 complexity class. That is, if $\mathbf{BFF}_2 = \mathbf{C}(\langle\beta_i\rangle)$, is there an $\alpha \in \mathbf{T_2TB}$ such that, $\mathbf{C}(\alpha) = \mathbf{C}(\langle\beta_i\rangle)$? We shall see that this is not the case in the next theorem.

Lemma 118 *Let $\langle\beta_i\rangle$ be useful. If there is an $\alpha \in \mathbf{T_2TB}$ such that $\mathbf{C_A}(\langle\beta_i\rangle) = \mathbf{C_A}(\alpha)$, then $\langle\beta_i\rangle$ is convergent.*

Proof: Since $\alpha \in \mathbf{T_2TB}$, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, there must exist $\sigma \subset f$ such that (σ, x) is a locking fragment for α. After α converges at (σ, x), if any β_i in the sequence has not converged, i.e., β_i allows an OTM_A $\widehat{M}_e$ to query beyond σ, then $\widehat{\varphi}_e \notin \mathbf{C_A}(\alpha)$.

This is because that any OTM_A queries outside σ may receive an arbitrary large oracle answer. If the length of the large answer exceeds $\alpha(\sigma, x)$, then the budget will be used up by reading the answer. □

If we allow $\langle\beta_i\rangle$ to be not useful, then Lemma 118, can be disproved by constructing a trivial $\langle\beta_i\rangle$, i.e., $\mathbf{C_A}(\beta_0) = \mathbf{C_A}(\beta_1) = \cdots$, in which each β_i delays its convergence until an inaccessible point is reached. Thus, no OTM_A clocked by any β_i can query the inaccessible point. In such a way, each β_i in the sequence determines the same complexity class and hence $\mathbf{C_A}(\beta_0) = \mathbf{C_A}(\langle\beta_i\rangle)$ but the convergence of $\langle\beta_i\rangle$ breaks if we choose a different inaccessible point for each β_i to converge.

The following lemma is a bit stronger than Lemma 118, but we can use the similar arguments to prove it and hence we omit the proof.

Lemma 119 *Let $\langle\beta_i\rangle$ be useful. If there is an $\alpha \in \mathbf{T_2TB}$ such that $\mathbf{C_A}(\langle\beta_i\rangle) \subseteq \mathbf{C_A}(\alpha)$, then $\langle\beta_i\rangle$ is convergent.* ■

Theorem 120 *$\mathbf{BFF}_2$ is not a type-2 complexity class.*

Proof: Suppose $\langle\beta_i\rangle = \mathbf{BFF}_2$. Without loss of generality, we can assume that $\langle\beta_i\rangle$ is useful. Thus, by Lemmas 118 and 111, $\mathbf{BFF}_2$ is not a type-2 complexity class. □

Instead of using Lemma 118, we can use Lemma 119 to prove a stronger result as:

Theorem 121 *There is no $\beta \in \mathbf{T_2TB}$ such that, $\mathbf{BFF}_2 \subseteq \mathbf{C_A}(\beta)$.* ■

These negative non-union results imply that a straightforward type-2 analog of the Union Theorem does not exist. In the next sub-section we show how to strengthen the hypotheses in order to have a type-2 Union Theorem under the answer-length-cost model.

6.1.3 The Union Theorem under The Answer-Length-Cost Model

McCreight and Meyer's original proof of the Union Theorem [MM69] is one of the first theorems in complexity theory proved by using a *finite injury priority argument*. We modify the proof and adopt the priority argument to prove our type-2 Union Theorem.

Recall Lemma 118 that, if $\langle\beta_i\rangle$ is useful and there is a $\alpha \in \mathbf{T_2TB}$ such that $\mathbf{C_A}(\alpha) = \mathbf{C_A}(\langle\beta_i\rangle)$, then $\langle\beta_i\rangle$ is convergent. Thus, the convergence of $\langle\beta_i\rangle$ is a necessary condition for claiming that $\mathbf{C}(\langle\beta_i\rangle)$ is a complexity class. On the other hand, Theorem 115 states that convergence together with uniformity, ascendancy, and usefulness are not sufficient to obtain a type-2 Union Theorem. Thus, we need to consider yet stronger hypotheses.

Theorem 122 (Type-2 Union Theorem) *Suppose that $\langle\beta_i\rangle$ is (i) uniform, (ii) ascending, (iii) useful, and (iv) strongly convergent. Then, there is an $\alpha \in \mathbf{T_2TB}$ such that, $\mathbf{C_A}(\alpha) = \mathbf{C_A}(\langle\beta_i\rangle)$.*

Proof: Let $\ell : \mathcal{F} \times \mathbf{N} \to \{0,1\}$ be a locking detector for $\langle\beta_i\rangle$ as in Definition 104. Let $\langle\cdot,\cdot\rangle : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ be our standard 1-1 pairing function with the extra property that: For all $x \in \mathbf{N}$ and $\sigma, \tau \in \mathcal{F}$, if $\sigma \subseteq \tau$, then $\langle\sigma, x\rangle \leq \langle\tau, x\rangle$. Such a $\langle\cdot,\cdot\rangle$ can be constructed easily.

The following requirements are naively lifted from the original arguments in the proof of the Union Theorem[1],

$$P_n^* : \overset{\infty}{\forall} (\sigma, x)[\beta_n(\sigma, x) \leq \alpha(\sigma, x)].$$

[1] See [Odi99], page 55

These would certainly be easy to satisfy, but they are not sufficient for our purposes. The problem is that, we have

$$\overset{\infty}{\forall}\ (\sigma, x)[\beta(\sigma, x) \leq \alpha(\sigma, x)] \ \not\Rightarrow\ \mathbf{C_A}(\beta) \subseteq \mathbf{C_A}(\alpha).$$

We therefore use the following requirements for our priority argument.

P_n**:** $\mathbf{C_A}(\beta_n) \subseteq \mathbf{C_A}(\alpha)$.

N_n**:** $[\widehat{\varphi}_n \notin \mathbf{C_A}(\langle\beta_i\rangle)] \ \Rightarrow\ [\widehat{\varphi}_n \notin \mathbf{C_A}(\alpha)]$.

The priority ordering of these is:

$$N_0 > P_0 > N_1 > P_1 > \cdots .$$

Clearly, if we can construct an $\alpha \in \mathbf{T_2TB}$ that satisfies all of these requirements, then the theorem follows.

We construct a type-2 time bound α in stages. We will make sure that every P_n will be satisfied after finitely many stages. Each N_n will be satisfied in the limit. That is, no finitary actions can guarantee that N_n is satisfied, but it will be so if we take certain actions infinitely often. Moreover, we will see in a moment how a P_i can be *injured* when we try to satisfy some N_j with $j \leq i$ in finitely many stages, although there are infinitely many stages in which some actions will be taken in order to satisfy each such N_j.

We maintain an array, *guess*, which is global to every stage. At the beginning of each stage $k > 0$, the values of $guess(0), guess(1), \ldots, guess(k-1)$ are natural numbers defined from the previous stage. Intuitively, $guess(e) = n$ means that we guess the functional $\widehat{\varphi}_e$ is in $\mathbf{C_A}(\beta_n)$. Each stage e will consider $\widehat{\varphi}$-programs $0, 1, \ldots, e$. In stage e, when $\widehat{\varphi}$-program e is first considered, we guess that $\widehat{\varphi}_e \in \mathbf{C_A}(\beta_e)$ and set $guess(e) = e$. In a later stage k, if we have evidence that $\widehat{\varphi}_e$ should not be in $\mathbf{C_A}(\beta_{guess(e)})$, then the value of $guess(e)$ will be changed to k.

Actions of The Algorithm for α: The algorithm for α is given in Figure 7 below. Here we discuss some of the strategies behind this algorithm. Recall Definition 103 that $\widetilde{\Phi}_e(\sigma, x)$ is the cost of computing $\widehat{\varphi}_e(\sigma^{\sim 0}, x)$ up to the completion of the first query outside $\mathsf{dom}(\sigma)$, or $\widetilde{\Phi}_e(\sigma, x)$ is simply the cost of computing $\widehat{\varphi}_e(\sigma^{\sim 0}, x)$ if no query outside $\mathsf{dom}(\sigma)$ made during the course of the computation.

In stage $k = \langle \sigma, x \rangle$ the algorithm for α takes σ and x as its inputs. If there exists $\tau \subset \sigma$ such that $\ell(\tau, x) = 1$, we force α to converge on (σ, x) and no further action will be taken. Otherwise, the algorithm checks programs $e = 0, 1, \ldots, k$ and takes some proper actions according to the following. If $\widetilde{\Phi}_e(\sigma, x) \leq \beta_{guess(e)}(\sigma, x)$, then this means that we don't have enough evidence to refute the current guess, $\beta_{guess(e)}$, for program e in this stage. In this case we shall let α provide enough budget for the computation of $\widehat{\varphi}_e(\sigma^{\sim 0}, x)$ to finish either completely or else up to the point where the first query outside $\mathsf{dom}(\sigma)$ is made. On the other hand, if $\beta_{guess(e)}(\sigma, x) < \widetilde{\Phi}_e(\sigma, x)$, then it is doubtful that $\widehat{\varphi}_e \in \mathbf{C_A}(\beta_{guess(e)})$. We thus force α to clip the computation of $\widehat{\varphi}_e(\sigma^{\sim 0}, x)$ and give program e a bigger guess, k, for another chance. For convenience, define:

$$
\begin{aligned}
A_k &= \left\{ e \;\middle|\; 0 \leq e \leq k, \widetilde{\Phi}_e(\sigma, x) \leq \beta_{guess(e)}(\sigma, x) \right\}. \\
B_k &= \left\{ e \;\middle|\; 0 \leq e \leq k, \beta_{guess(e)}(\sigma, x) < \widetilde{\Phi}_e(\sigma, x) \right\}.
\end{aligned}
$$

We use **A1** and **A2** to denote the following two actions.

A1: α is defined so as to provide enough budget allowing the computations of the $\widehat{\varphi}$-programs in A_k on $(\sigma^{\sim 0}, x)$ to finish either completely or else up to the point where the first query outside $\mathsf{dom}(\sigma)$ is made. This action explicitly tries to satisfy requirements $P_{guess(0)}, P_{guess(1)}, \ldots$, and $P_{guess(k)}$.

A2: α is defined so as to clip the computations of the $\widehat{\varphi}$-programs in B_k on $(\sigma^{\sim 0}, x)$,

```
Program for α
    input   σ : F, x : N;
    k ⟵ ⟨σ, x⟩;                                    /* We call this stage k.   */
/* guess is an array of natural numbers global to all stages. */
/* The initial values of guess(0),guess(1),...,guess(k − 1) for this stage k
   are obtained by running α on every (σ′, x′) with 0 ≤ ⟨σ′, x′⟩ < k. */
    for ⟨σ′, x′⟩ = 0 to k − 1 Run α(σ′, x′);

/* Guess that the new φ̂-program k is bounded by β_k. */
    guess(k) ⟵ k;

/* We check if α has to converge at this point. */
△   if ∃τ[τ ⊂ σ and ℓ(τ, x) = 1)]
        then return max({α(τ, x)|τ ⊂ σ});

/* A1: Since max(guess(0), guess(1),..., guess(k − 1)) < k and ⟨β_i⟩ is as-
   cending, we set ceiling to β_k(σ, x) as the maximal value for α(σ, a) in such
   a way the action A1 can be satisfied. */
    ceiling ⟵ β_k(σ, x);

/* A2: In the following, we check φ̂-programs e = 0, 1,..., k. If φ̂_e is not
   bounded by β_guess(e) on (σ, x), then we take the action A2 by lowering
   ceiling and assign the new guess, k, for e.                                */
    for e = 0 to k do
        if Φ̃_e(σ, x) > β_guess(e)(σ, x) do              /* e is picked on.   */
            ceiling ⟵ min(ceiling, β_guess(e)(σ, x));
            guess(e) ⟵ k;
        end if-do;
    end for-do;
    return ceiling;
End program
```

Figure 7: An Algorithm for The Type-2 Union Theorem

and the algorithm will give these programs a new guess, k. This action is intended to satisfy requirements $N_0, N_1, \ldots, N_k$ but it may also injure some $P_{guess(e)}$, where $e \in A_k$.

We shall argue in a moment that every requirement will be satisfied eventually as the stages progressed. In other words, each P_e can be injured by some N_k at most finitely many times. Note that, for every $n \in \mathbf{N}$, $P_{n+1} \Rightarrow P_n$. Thus, if P_k is satisfied, so is every P_n with $n \leq k$.

Terminology and Notation: Suppose $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$ and $\langle \sigma, x \rangle = k$.

1. Let $guess_k$ denote the state of the array, $guess$, as of the beginning of stage k. The variable $ceiling : \mathbf{N}$ that is local to every stage holds the maximal value of $\alpha(\sigma, x)$ for the action **A2**. Let $ceiling_k$ denote the value of $ceiling$ as of the end of stage k. Thus, $\alpha(\sigma, x) = ceiling_k$.

2. We say that the $\widehat{\varphi}$-program $e \leq k$ ***participates*** in lowering $ceiling$ in stage k if and only if $\widetilde{\Phi}_e(\sigma, x) > \beta_{guess_k(e)}(\sigma, x)$ and, for all $\tau \subset \sigma$, $\ell(\tau, x) = 0$.

3. We say that α ***picks on*** $\widehat{\varphi}$-program e at (σ, x) if and only if

$$0 \leq e \leq k \text{ and } \widetilde{\Phi}_e(\sigma, x) > \alpha(\sigma, x).$$

According to the definition of $\widetilde{\Phi}_e$, for every $\sigma, \tau \in \mathcal{F}$ with $\sigma \subset \tau$ and $x \in \mathbf{N}$, we have $\widetilde{\Phi}_e(\sigma, x) \leq \widetilde{\Phi}_e(\tau, x)$. It is clear that, if α *picks on* $\widehat{\varphi}$-program e at (σ, x), then, for all $f \supset \sigma$, the clock α will clip the computation of $\widehat{M}_e$ on (f, x), i.e., $\widehat{\varphi}_{e,\alpha}(f, x) \Uparrow$. However, since (σ, x) is not necessarily a locking fragment of $\widehat{\varphi}_e$ and it is possible that $\alpha(\sigma, x) \leq \alpha(\tau, x)$, it follows that there may exist some other $\widehat{\varphi}$-program e' for $\widehat{\varphi}_e$ such that, at the beginning of the computation, $\widehat{\varphi}$-program e' queries the oracle on $\mathsf{dom}(\tau)$ in order to gain the budget $\alpha(\tau, x)$ first. In such a way, we may have $\widehat{\varphi}_{e',\alpha}(f, x) \Downarrow$.

4. We say that $\widehat{\varphi}$-program e is a ***casualty*** of α at (σ, x) in stage k, if and only if there exists some $i \leq k$ such that

$$\begin{aligned} \beta_{guess_k(i)}(\sigma, x) &< \widetilde{\Phi}_i(\sigma, x), \text{ and} \\ \beta_{guess_k(i)}(\sigma, x) &< \widetilde{\Phi}_e(\sigma, x) \ \leq \ \beta_{guess_k(e)}(\sigma, x). \end{aligned}$$

On (σ, x), α does not intend to *pick on* e, but when α tries to *pick on* some other $\widehat{\varphi}$-program (in the case above, $\widehat{\varphi}$-program i), it lowers *ceiling* enough so that the clock α may clip some computations of $\widehat{M}_e$ on (f, x) as a casualty, where $\sigma \subset f$.

Unlike the construction for the original proof of the Union Theorem [MM69, Odi99], in some cases of $\widehat{\varphi}_e \in \mathbf{C}_{\mathbf{A}}(\langle \beta_i \rangle)$, the value of *guess*$(e)$ may change infinitely often due to the fact that a compact set may be infinite. The difference between $\widehat{\varphi}_e \in \mathbf{C}_{\mathbf{A}}(\langle \beta_i \rangle)$ and $\widehat{\varphi}_e \notin \mathbf{C}_{\mathbf{A}}(\langle \beta_i \rangle)$ is the following. In case that $\widehat{\varphi}_e \in \mathbf{C}_{\mathbf{A}}(\langle \beta_i \rangle)$, if α should change its guess for program e on infinitely many (σ, x)'s, then all but finitely many of these are locking fragments of $\widehat{\varphi}_e$ but *not minimal ones*. That is, the set $(\!(\sigma, x)\!)$ is not $\mathbb{T}(\widehat{\varphi}_e)$-open, and there are finitely many basic $\mathbb{T}(\widehat{\varphi}_e)$-open sets that cover all of such infinitely many $(\!(\sigma, x)\!)$'s. On the other hand, if $\widehat{\varphi}_e \notin \mathbf{C}_{\mathbf{A}}(\langle \beta_i \rangle)$, α will change its guess on infinitely many minimal locking fragments of $\widehat{\varphi}_e$.

Claim 1: $\alpha \in \mathbf{T}_2\mathbf{TB}$. We shall prove that α is computable, nontrivial, $\mathcal{F}$-monotone, and convergent.

1. Computability. Since $\langle \beta_i \rangle$ is strongly convergent, the locking detector ℓ is recursive. Also, it is clear that $\widetilde{\Phi}_e(\sigma, x) > \beta_{guess(e)}(\sigma, x)$ is recursively decidable. Thus, α is recursively computable.
2. Nontriviality. For any (σ, x), the value of $\alpha(\sigma, x)$ is based on the value of $\beta_i(\sigma, x)$ for some $i \in \mathbf{N}$. Thus, α must be nontrivial.

3. $\mathcal{F}$-monotonicity. Since whenever α decides to converge on (σ, x), α outputs $\mathsf{max}(\{\alpha(\tau, x) | \tau \subset \sigma\})$, it is clear that α is $\mathcal{F}$-monotonicity.

4. Convergence. Fix $(f, x) \in \mathcal{T} \times \mathbf{N}$. Since $\langle \beta_i \rangle$ is strongly convergent and ℓ is its locking detector, there is a $\sigma \in \mathcal{F}$ with $\sigma \subset f$ such that the condition of the if statement in line marked $\triangle$ is true. Thus, for all $\tau \supseteq \sigma$, $\alpha(\tau, x) = \alpha(\sigma, x)$ and hence α is convergent.

Therefore, $\alpha \in \mathbf{T}_2\mathbf{TB}$.

Note that although we have the assumption that $\langle \beta_i \rangle$ is useful, α may not be useful because it is possible that $\ell(\sigma, x) = 0$ and $\beta_i(\sigma, x) \downarrow$, and

$$\alpha(\sigma, x) = \beta_i(\sigma, x) < \|\sigma\| + |x| + 2.$$

We will see later that this introduces a difficulty when we need to patch some $\widehat{\varphi}$-program e that is clipped by α at a point (σ, x) in case (σ, x) is not a locking fragment of α.

Claim 2: $\bigcup_{n=0}^{\infty} \mathbf{C}_{\mathbf{A}}(\beta_n) \subseteq \mathbf{C}_{\mathbf{A}}(\alpha)$.

Fix $n \in \mathbf{N}$ and suppose that $F \in \mathbf{C}_{\mathbf{A}}(\beta_n)$. Moreover, suppose that e is a $\widehat{\varphi}$-program for F and E_{e,β_n} is compact in $\mathbb{T}(\widehat{\varphi}_e)$. We shall prove that $F \in \mathbf{C}_{\mathbf{A}}(\alpha)$. Note that we cannot prove that $E_{e,\alpha}$ is compact in $\mathbb{T}(\widehat{\varphi}_e)$. Instead, we will prove that there is another $\widehat{\varphi}$-program p for F such that $E_{p,\alpha}$ is compact in $\mathbb{T}(\widehat{\varphi}_p)$. Define:

$$I_e = \left\{ (\sigma, x) \in \mathcal{F} \times \mathbf{N} \;\middle|\; \alpha(\sigma, x) < \widetilde{\Phi}_e(\sigma, x) \right\}. \tag{41}$$

$$O_e = \left\{ (\sigma, x) \in \mathcal{F} \times \mathbf{N} \;\middle|\; (\!(\sigma, x)\!) \text{ is open in } \mathbb{T}(\widehat{\varphi}_e) \right\}. \tag{42}$$

$$X_e = I_e \cap O_e. \tag{43}$$

I_e is the set on which $\widehat{\varphi}$-program e is picked on by α. O_e is the set of the minimal locking fragments of $\widehat{\varphi}_e$ and the sub-fragments of the minimal locking fragments of $\widehat{\varphi}_e$. In other words, for every $(\sigma, x) \in O_e$, there exists some $\tau \supseteq \sigma$ such that, $(\tau, x) \in O_e$ and (τ, x) is a minimal locking fragment of $\widehat{\varphi}_e$. (See Definition 30 on page 57 for the topology $\mathbb{T}_{min}(\widehat{\varphi}_e)$.)

We first prove that X_e is finite. Then, we show that there is another $\widehat{\varphi}$-program p for F such that $E_{p,\alpha}$ is compact in $\mathbb{T}(\widehat{\varphi}_p)$. Note that if p is another $\widehat{\varphi}$-program for F, then, by definition, $\mathbb{T}(\widehat{\varphi}_p) = \mathbb{T}(\widehat{\varphi}_e) = \mathbb{T}(F)$, although $\widehat{\varphi}$-programs p and e may make different queries. We will carefully distinguish between $\mathbb{T}(\widehat{\varphi}_p)$ and $\mathbb{T}(\widehat{\varphi}_e)$ when talking about the compactness of $E_{p,\alpha}$ or $E_{e,\alpha}$ in order to avoid confusion.

Claim 2.1: X_e is finite.

In any stage we say that the guess for e is a *bad guess* if $guess(e) < n$. Also, we say that (f, x) is a *bad input* if $(f, x) \in E_{e,\beta_n}$. Without loss of generality, we may assume that, if $(f, x) \in E_{e,\beta_n}$, then for every $i \in \mathbf{N}$ $\widehat{\varphi}_{e,\beta_i}(f, x) \Uparrow$.

Fix any $(\sigma, x) \in O_e$. In the course of the computation of α on (σ, x), there are three possible reasons for (σ, x) to be in I_e: (i) the ceiling is too low due to a bad guess from some previous stage for e, (ii) e is a casualty when α tries to pick on some other $\widehat{\varphi}$-programs at (σ, x), and (iii) $\sigma \subset f$ and (f, x) is a bad input, i.e., $(f, x) \in E_{e,\beta_n}$. Note that these three reasons are not exclusive.

1. **Bad Guess:** According to the algorithm, whenever e participates in lowering *ceiling*, the value of $guess(e)$ will be changed to a bigger one. Thus, only a finite number of bad guesses could have been made before $guess(e)$ becomes equal to or bigger than n. Therefore, only finitely many (σ, x)'s could be introduced into X_e due to bad guesses.

2. **Casualty:** For some $e' \in \mathbf{N}$, α tries to pick on the $\widehat{\varphi}$-program e' at (σ, x) by lowering *ceiling* in stages $\langle \sigma, x \rangle$, but the ceiling is too low so that the computation of $\widehat{\varphi}$-program e will be clipped by α as a casualty. One can see that such e' cannot be bigger than n, otherwise the guess for e' is always bigger than n and the ceiling will never get too low to for e. There are only finitely many such $\widehat{\varphi}$-programs with indices smaller that n. For each such e', it does not matter whether $\widehat{\varphi}_{e'}$ is in $\mathbf{C_A}(\langle \beta_i \rangle)$, the guess for e' will become bigger and bigger until it gets a correct one or passes n. Moreover, since $\langle \beta_i \rangle$ is ascending, it follows that this $\widehat{\varphi}$-program e' introduces at most finitely many (σ, x)'s into X_e.
3. **Bad Inputs:** Suppose $(\sigma, x) \in X_e$ not because of the two reasons discussed above. It must be the case that $\sigma \subset f$ for some $(f, x) \in E_{e,\beta_n}$. One can see that (σ, x) must be a minimal locking fragment of $\widehat{\varphi}_e$, otherwise E_{e,β_n} cannot be compact in $\mathbb{T}(\widehat{\varphi}_e)$. Thus, $(\!(\sigma, x)\!)$ is a basic open set of E_{e,β_n}. By the assumption that E_{e,β_n} is compact in $\mathbb{T}(\widehat{\varphi}_e)$, we only have finitely many such (σ, x).

Therefore, X_e is finite. Next, we shall argue that, $\widehat{\varphi}_e \in \mathbf{C_A}(\alpha)$.

Claim 2.2: $\widehat{\varphi}_e \in \mathbf{C_A}(\alpha)$.

It is clear that if we clock e with α we have

$$E_{e,\alpha} = \left\{ (\!(\sigma, x)\!) \;\middle|\; (\sigma, x) \in X_e \right\}.$$

However, $E_{e,\alpha}$ may not be compact in $\mathbb{T}(\widehat{\varphi}_e)$ because there may be some $(\sigma, x) \in X_e$ that is not a locking fragment of $\widehat{\varphi}_e$ due to a bad guess or casualty as discussed above. Let

$$P_e = \left\{ (\sigma, x) \;\middle|\; (\sigma, x) \in X_e \text{ and } (\sigma, x) \text{ is not a locking fragment of } \widehat{\varphi}_e. \right\}$$

Our goal is to show that we can patch $\widehat{\varphi}$-program e on those $(\sigma, x) \in P_e$ under the budget provided by α.

At first, we show that, for every $(\sigma, x) \in P_e$,

$$\alpha(\sigma, x) \geq \|\sigma\| + |x| + 2.$$

Note that α may not be useful. Suppose $(\sigma, x) \in P_e$ and, for some $i \in \mathbf{N}$,

$$\beta_i(\sigma, x) = \alpha(\sigma, x).$$

Since (σ, x) is not a locking fragment of $\widehat{\varphi}_e$, it follows that, for every $f \supset \sigma$, $\widehat{\varphi}$-program e on (f, x) must query the oracle outside $\mathsf{dom}(\sigma)$. Thus, β_n cannot converge at (σ, x), otherwise E_{e,β_n} is not compact in $\mathbb{T}(\widehat{\varphi}_e)$ under the answer-length-cost model. By the assumptions that $\langle \beta_i \rangle$ is uniformly convergent and that β_n does not converge at (σ, x), we know that neither does β_i converge at (σ, x). Since β_i is useful, it follows that

$$\beta_i(\sigma, x) = \alpha(\sigma, x) \geq \|\sigma\| + |x| + 2.$$

Similarly, for every $\eta \subseteq \sigma$, there is some $j \in \mathbf{N}$ such that,

$$\beta_j(\eta, x) = \alpha(\eta, x) \geq \|\eta\| + |x| + 2. \tag{44}$$

Let q be the first query outside $\mathsf{dom}(\sigma)$ made during the course of the computation of $\widetilde{\Phi}_e(\sigma, x)$. Clearly, for all $\tau \in \mathcal{F}$ such that $\mathsf{dom}(\tau) - \mathsf{dom}(\sigma) = \{q\}$, we have $(\tau, x) \in O_e$. Suppose $(\tau, x) \notin I_e$ and $\langle \tau, x \rangle = k$. We have

$$\widetilde{\Phi}_e(\tau, x) \leq \beta_{guess_k(e)}(\tau, x) \leq ceiling_k = \alpha(\tau, x).$$

Thus, we can have a $\widehat{\varphi}$-program e' to check $\sigma \subset f$ at the beginning of the computation on any $(f, x) \in \mathcal{T} \times \mathbf{N}$. If $\sigma \subset f$ is false, e' computes $F(f, x)$

exactly same as e does. If $\sigma \subset f$ is true, e' queries $[f(q) =?]$. This can be done by an $\mathrm{OTM}_{\mathrm{A}}$ clocked by α because of (44), and in case the extra query $[f(q) =?]$ is needed, the budget, $\alpha(\sigma, x)$, must be enough to make the extra query because β_i does not converge at (σ, x) and β_i is useful. It is clear that

$$X_{e'} = X_e - \{(\sigma, x)\}.$$

Since P_e is finite, we can use the same idea to remove every $(\sigma, x) \in P_e$ and result in a patched $\widehat{\varphi}$-program p for $\widehat{\varphi}_e$ such that,

$$E_{p,\alpha} = \left\{ (\!(\sigma, x)\!) \;\middle|\; (\sigma, x) \in (X_e - P_e) \right\}.$$

Clearly, $E_{p,\alpha}$ is compact in $\mathbb{T}(\widehat{\varphi}_p)$. Since $F = \widehat{\varphi}_e = \widehat{\varphi}_p$, it follows that $F \in \mathbf{C_A}(\alpha)$.

Claim 3: $\mathbf{C_A}(\alpha) \subseteq \bigcup\limits_{n=o}^{\infty} \mathbf{C_A}(\beta_n)$.

Suppose $\widehat{\varphi}_n \notin \mathbf{C_A}(\langle\beta_i\rangle)$. It is easy to see that any guess for $\widehat{\varphi}$-program n cannot satisfy n forever, and thus α will lower down the ceiling to pick on $\widehat{\varphi}$-program n at infinitely many locking fragments of $\widehat{\varphi}_n$. Since X_n is not finite, we cannot patch $\widehat{\varphi}$-program n under a fixed cost in the same way discussed in Claim 2.2. Thus, $\widehat{\varphi}_n \notin \mathbf{C_A}(\alpha)$.

Since $\mathbf{C_A}(\langle\beta_i\rangle) = \bigcup\limits_{n=o}^{\infty} \mathbf{C_A}(\beta_n)$, by Claims 2 and 3, we therefore have $\mathbf{C_A}(\alpha) = \mathbf{C_A}(\langle\beta_i\rangle)$ □

Discussion 1: The *uniform convergence* of a $\langle\beta_i\rangle$ is indeed a very strong condition (so is *strong convergence*) in the sense that, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, every β_i in the sequence has to refer to the same fragment of f. We do not see any reasonable way to take the liberty of letting each β_i make oracle queries independently. We observe that the two sample $\langle\beta_i\rangle$'s constructed in the proof of the Type-2 Non-Union Theorem (Theorem 115, page 143) and in the big-O notation (Corollary 126, page 161) are not bounded, i.e., $\lim_{i\to\infty} F_{\beta_i}(f,x) = \infty$. We may want to, for example, have $\langle\beta_i\rangle$ bounded by some continuous functional in order to relax the *uniform convergence* as a hypothesis of the type-2 Union Theorem. The next corollary denies this hope.

Corollary 123 *There exist a continuous functional $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ and a uniform, ascending, and useful $\langle\beta_i\rangle$ such that, for every $i \in \mathbf{N}$, $F_{\beta_i} \leq F$, and $\mathbf{C_A}(\langle\beta_i\rangle)$ is not a type-2 complexity class.*

Sketch of Proof: Note that the $\langle\beta_i\rangle$ given in the proof of Theorem 115 is not bounded. We modify it as follows: let $d \geq \mathsf{max}(c, c', a)$, where c, c', and a are the same constants in the arguments of the proof of Theorem 115. Define

$$\beta_i(\sigma, x) = \begin{cases} |x| + c & \text{if } x \geq i; \\ \|\sigma_{[x+2]}\| + |x| + d & \text{otherwise.} \end{cases}$$

It is clear that, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$ and $i \in \mathbf{N}$, β_i is bounded as

$$F_{\beta_i}(f, x) \leq F(f, x) = \|f_{[x+2]}\| + |x| + d.$$

Then, we use the exact arguments in the proof of Theorem 115 except the definition for $\langle\beta_i\rangle$ to prove this corollary. □

Note that if $\langle\beta_i\rangle$ is bounded by a total continuous functional, then, by Lemma 106, $\langle\beta_i\rangle$ is convergent but not necessarily uniformly convergent.

Discussion 2: Recall Definition 42 on page 71 that a *strong* type-2 time bound, β, is a type-2 time bound which is strongly $\mathcal{F}$-monotone, i.e., for every $\sigma, \tau \in \mathcal{F}$ and $a \in \mathbf{N}$,

$$\sigma \subseteq \tau \Rightarrow \beta(\sigma, a) \leq \beta(\tau, a).$$

We say that $\langle\beta_i\rangle$ is strong if and only if every β_i in $\langle\beta_i\rangle$ is strongly $\mathcal{F}$-monotone.

Computations clocked with such kind of time bounds have an intuitive advantage that the budget provided by the clock will never shrink during the courses of the computations. Thus, we may want the type-2 time bound α constructed in the proof of the type-2 Union Theorem to be strong. At least, we hope so in case that $\langle\beta_i\rangle$ itself is strong.

Consider the proof of Theorem 122 (the Type-2 Union Theorem). The construction for α does not guarantee that α is strongly $\mathcal{F}$-monotone. It is possible that α shrinks on some $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$ in order to pick on some $\widehat{\varphi}$-program e as in the following case. Suppose $guess(e)$ contains an index i such that $\widehat{\varphi}_e \notin \mathbf{C_A}(\beta_i)$ and α needs to pick on e on input (σ, a). Moreover, on every (τ, a) with $\tau \subset \sigma$, α does not pick on e because e queries outside $\mathsf{dom}(\tau)$ and β_i's budget is enough for e to complete the queries. Thus, it is possible that α on some (τ, a) with $\tau \subset \sigma$ outputs a value bigger than $\beta_i(\sigma, a)$.

Thus, we need to explicitly maintain a floor in order to prevent α from shrinking. However, such floor creates a difficulty in trying to prove $\mathbf{C_A}(\alpha) \subseteq \mathbf{C_A}(\langle\beta_i\rangle)$. This is because we have to argue that: If $F \notin \mathbf{C_A}(\langle\beta_i\rangle)$, then there is no $\widehat{\varphi}$-program e for F such that, every time when α tries to pick on e, e is protected by the floor. That is, we need to prove that there is no such $\widehat{\varphi}$-program e for F such that, for every stage $k = \langle\sigma, x\rangle$ when α tries to pick on e at (σ, x), the following situation occurs:

$$\text{floor} \geq \widetilde{\Phi}_e(\sigma, x) > \beta_{guess(e)}(\sigma, x).$$

At the moment we do not know a way to fix this problem when we try to construct a strong α in the proof of the Type-2 Union Theorem. We have the following conjecture.

Conjecture 124 *There is a uniform, ascending, and strong $\langle \beta_i \rangle$ such that, if there is $\alpha \in \mathbf{T_2TB}$ such that $\mathbf{C_A}(\alpha) = \mathbf{C_A}(\langle \beta_i \rangle)$, then α is not strong.* ■

6.1.4 The Type-2 Big-O Notation

The big-O notation is a key tool in algorithm analysis. We define a type-2 analog of the big-O notation as follows.

Definition 125 (Type-2 big-O Notation) *Given $\beta \in \mathbf{T_2TB}$, define*

$$\mathbf{O}(\beta) = \left\{ \widehat{\varphi}_e \;\middle|\; \widehat{\varphi}_e \in \mathbf{C_A}(c\beta + d) \text{ for some } c, d \in \mathbf{N} \right\}.$$

■

One of the prime motivations for proving the Type-2 Union Theorem is to have $\mathbf{O}(\beta)$ as a type-2 complexity class. Since we need some restrictions on β in order to apply the Type-2 Union Theorem, we do not know if $\mathbf{O}(\beta)$ is a type-2 complexity class for every $\beta \in \mathbf{T_2TB}$. However, the restrictions of the type-2 Union Theorem become acceptable when we consider the classes given by the big-O notation, since the strongest hypothesis in the type-2 Union Theorem, *strong convergence*, automatically holds if the β in $\mathbf{O}(\beta)$ is locking detectable. From Theorem 122, we have an immediate corollary:

Corollary 126 *Let $\beta \in \mathbf{T_2TB}$. If β is locking detectable and useful, then there is an $\alpha \in \mathbf{T_2TB}$ such that $\mathbf{C_A}(\alpha) = \mathbf{O}(\beta)$.*

Proof: Let $c, d \in \mathbf{N}$ and $i = \mathsf{max}(c, d)$. By Theorem 67,

$$\mathbf{C_A}(c\beta + d) \subseteq \mathbf{C_A}(i\beta + i).$$

Thus, it is clear that

$$\mathbf{O}(\beta) = \mathbf{C_A}(\langle\beta_i\rangle), \text{ where } \beta_i = i\beta + i.$$

It is clear that $\langle\beta_i\rangle$ is uniform and ascending. Suppose β is locking detectable and useful. Then, for every $i > 0$, $i\beta + i$ is also useful. Moreover, if ℓ is a locking detector for β, then ℓ is a locking detector for $\langle\beta_i\rangle$ as well. Thus, $\langle\beta_i\rangle$ is strongly convergent. By Theorem 122, there is an $\alpha \in \mathbf{T_2TB}$ such that $\mathbf{C_A}(\alpha) = \mathbf{C_A}(\langle\beta_i\rangle) = \mathbf{O}(\beta)$. □

Note that, although we have Theorem 88 asserting that we can have an effective operator Θ_L such that, $\Theta_L(\beta)$ is locking detectable and equivalent to β, but

$$[\mathbf{C_A}(\beta) = \mathbf{C_A}(\Theta_L(\beta))] \not\Rightarrow [\mathbf{C_A}(i\beta + i) = \mathbf{C_A}(i\Theta_L(\beta) + i)].$$

On the other hand, if we define $\beta_i = \Theta_L(i\beta + i)$, the strong convergence of $\langle\beta_i\rangle$ may not hold. This is because, if $i \neq j$, the inaccessible points of β_i and β_j are different. Thus, we cannot remove the hypothesis that β must be locking detectable in Corollary 126.

Corollary 127 *Let $\alpha, \beta \in \mathbf{T_2TB}$. If α and β are locking detectable and useful, then $\mathbf{O}(\alpha + \beta)$ is a type-2 complexity class.*

Sketch of Proof: Let ℓ_α, ℓ_β be locking detectors of α and β, respectively. For every $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, define

$$\gamma(\sigma, x) = \alpha(\sigma, x) + \beta(\sigma, x).$$

Clearly, γ is a useful type-2 time bound if both α and β are. For locking detectability, define $\ell : \mathcal{F} \times \mathbf{N} \to \{0, 1\}$ by

$$\ell(\sigma, x) = \begin{cases} 1 & \text{if } \ell_\alpha(\sigma, x) = \ell_\beta(\sigma, x) = 1; \\ 0 & \text{otherwise.} \end{cases}$$

Thus, ℓ is a locking detector for γ. By Corollary 126, $\mathbf{O}(\alpha+\beta)$ is a type-2 complexity class. $\square$

The following corollary states that we can drop the less significant term in the big-O notation.

Corollary 128 *Let $\alpha, \beta \in \mathbf{T}_2\mathbf{TB}$. Suppose that both α and β are locking detectable and useful. If $\alpha \leq^* \beta$, then $\mathbf{O}(\alpha + \beta) = \mathbf{O}(\beta)$.*

Proof: We skip the proof which can be obtained straightforwardly. $\square$

We will prove in next sub-section that $\mathbf{O}(\beta)$ under the unit-cost model is not a type-2 complexity class. Thus, we do not consider it is of interest to define the big-O notation under the unit-cost model.

6.1.5 The Non-Union Theorem under The Unit-Cost Model

Recall the type-2 Union Theorem under the answer-length-cost model and consider the algorithm for α in Figure 7 on page 151. Since $\langle\beta_i\rangle$ is strongly convergent, it is easy for our program for α to decide the right moment to converge based on the value of the locking detector of $\langle\beta_i\rangle$, which indicates that every β_i has converged. After α converges, any $\widehat{\varphi}$-program e queries outside the domain of the minimal locking fragment of α cannot be in $\mathbf{C_A}(\alpha)$ nor the union class $\mathbf{C_A}(\langle\beta_i\rangle)$. Under the unit-cost model, however, we cannot make use of this advantage of the strong convergence of $\langle\beta_i\rangle$. This is because when all β_i's and α have locked on (σ, x), there always exists some $\mathrm{OTM_U}$ that queries information beyond σ at a fixed expense. Even if the fixed expense is greater than $\alpha(\sigma, x)$, it is possible that for some sufficiently large i, the budget provided by β_i can suppose the expense, and thus $\widehat{\varphi}_e \in \mathbf{C_A}(\langle\beta_i\rangle) - \mathbf{C_A}(\alpha)$. Consider the functionals defined in the following. Under the unit-cost model, each of them can be computed under a fixed amount of cost plus the cost of scanning the type-0 input.

Definition 129 *Let $k \in \mathbf{N}$ and $(\tau, a) \in \mathcal{F} \times \mathbf{N}$. Define $G^k_{\tau,a} : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ by*

$$G^k_{\tau,a}(f,x) = \begin{cases} 2^{k+1} & \textit{if } x = a, \tau \subset f, \textit{ and } f(\mathsf{max}(\mathsf{dom}(\tau)) + 1) \textit{ is odd}, \\ 0 & \textit{otherwise.} \end{cases}$$

The following theorem shows an easy example that the type-2 analog of the big-O notation, $\mathbf{O}(\beta)$, is not a complexity class under the unit-cost model. The theorem also implies that the strong convergence of $\langle \beta_i \rangle$ does not provide a sufficient condition to yield a Union Theorem under the unit-cost model.

Theorem 130 *Given any $\beta \in \mathbf{T_2TB}$,*

$$\forall \alpha \in \mathbf{T_2TB}, \mathbf{C_U}(\alpha) \neq \bigcup_{c \in \mathbf{N}} \mathbf{C_U}(c\beta).$$

Proof: Fix any two $\alpha, \beta \in \mathbf{T_2TB}$ and a $(g, a) \in \mathcal{T} \times \mathbf{N}$. Let $\tau \subset g$. Suppose that (τ, a) is a minimal locking fragment of α and $\alpha(\tau, a) = k$. Consider the functional $G^k_{\tau,a}$ defined in Definition 129. Let $d = \mathsf{max}(\mathsf{dom}(\tau)) + 1$.

Claim 1 : $G^k_{\tau,a} \notin \mathbf{C_U}(\alpha)$.

For each $i \in \mathbf{N}$, define

$$\sigma_i = \tau \cup \{(d, 2i + 1)\}.$$

For each $i \in \mathbf{N}$ and $f \in \mathcal{T}$, if $\sigma_i \subset f$, then any $\widehat{\varphi}_e$ that computes $G^k_{\tau,a}$ will be clipped by α during the course of the computation on (f, a). Let $X = \bigcup_{i \in \mathbf{N}} (\!(\sigma_i, a)\!)$. Clearly, $X \subseteq E_{e,\alpha}$. Moreover, X is not compact in $\mathbb{T}(\widehat{\varphi}_e)$ because

$$\mathcal{O} = \{(\!(\sigma_0, a)\!), (\!(\sigma_1, a)\!), \dots\},$$

is an open cover for X without finite subcover. Since $X \subseteq E_{e,\alpha}$, by Lemma 34, $E_{e,\alpha}$ is not compact in $\mathbb{T}(\widehat{\varphi}_e)$. Therefore, $G^k_{\tau,a} \notin \mathbf{C_U}(\alpha)$.

Claim 2 : $G^k_{\tau,a} \in \bigcup_{c\in\mathbf{N}} \mathbf{C}_{\mathbf{U}}(c\beta)$.

We construct an $\mathrm{OTM}_{\mathrm{U}}$, $\widehat{M}^u_e$, that computes $G^k_{\tau,a}$ as follows: On any input $(f,x) \in \mathcal{T} \times \mathbf{N}$, the machine will, at first, (i) scan x and check if $(x = a)$, then, (ii) check if $\tau \subset f$, (iii) check if $f(d)$ is odd, and (iv) output 0 or 2^{k+1} based on the results in (i), (ii), and (iii).

Under the unit-cost model, the cost for an $\mathrm{OTM}_{\mathrm{U}}$ to compare $\tau(i)$ and $f(i)$, where $f \in \mathcal{T}$ and $i \in \mathsf{dom}(\tau)$, is bounded by $c(|i| + |\tau(i)|)$ for some constant $c > 1$, because the $\mathrm{OTM}_{\mathrm{U}}$ only has to scan at most $2 \cdot |\tau(i)|$ many bits to judge if the two are identical. Note that in the case that $|f(i)| > |\tau(i)|$, $\widehat{M}^u_e$ does not have to scan all of $f(i)$. Likewise, the machine can just scan the least significant bit of $f(d)$ to know whether $f(d)$ is odd. Thus, the cost of the computation of $\widehat{M}^u_e$ on any (f,x) is bounded as follows:

$$\widehat{\Phi}^u_e(f,x) \leq c(|x| + ||\tau|| + |d| + k + 1).$$

Since $||\tau|| + |d| + k + 1$ is also a constant, there is a constant $c' \in \mathbf{N}$ such that,

$$\widehat{\Phi}^u_e(f,x) \leq c'|x| + c'.$$

Since β is a type-2 time bound, which is nontrivial, it follows that, for every $(\sigma,x) \in \mathcal{F} \times \mathbf{N}$, $\beta(\sigma,x) \geq |x| + 1$, and thus,

$$c'\beta(\sigma,x) \geq c'|x| + c'.$$

That is, if $\widehat{M}^u_e$ is clocked by $c'\beta$, $\widehat{M}^u_e$ will get enough budget from $c'\beta$ during the course of the computation on every (f,x). Thus, $G^k_{\tau,a} \in \mathbf{C}_{\mathbf{U}}(c'\beta)$.

Therefore, from Claim 1 and Claim 2, $G^k_{\tau,a} \in \bigcup_{c\in\mathbf{N}} \mathbf{C}_{\mathbf{U}}(c\beta) - \mathbf{C}_{\mathbf{U}}(\alpha)$, and, hence, $\mathbf{C}_{\mathbf{U}}(\alpha) \neq \bigcup_{c\in\mathbf{N}} \mathbf{C}_{\mathbf{U}}(c\beta)$. $\square$

Corollary 131 *For any $\beta \in \mathbf{T}_2\mathbf{TB}$, $\mathbf{O}(\beta)$ is not a complexity under the unit-cost model.*

Proof: We immediately have this corollary from Theorem 130 $\square$

The following corollary is a stronger analog of Corollary 123 in the sense that we restrict $\langle\beta_i\rangle$ to be strongly convergent but we still do not have a type-2 Union Theorem under the given conditions and the unit-cost model computations. We conclude from this that we need stronger hypotheses to have a type-2 union theorem under the unit-cost model.

Corollary 132 *There exist a total computable functional $H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ and a uniform, ascending, useful, and strongly convergent $\langle\beta_i\rangle$ such that, for every $i \in \mathbf{N}$, $F_{\beta_i} \leq_{\mathbf{2}}^{*} H$ and $\mathbf{C_U}(\langle\beta_i\rangle)$ is not a type-2 complexity class.*

Proof: Fix any useful and locking detectable $\beta \in \mathbf{T_2TB}$. Let $\beta_i = i\beta$ for every $i \in \mathbf{N}$. Clearly, such $\langle\beta_i\rangle$ is uniform, ascending, useful, and strongly convergent. By Theorem 130, $\mathbf{C_U}(\langle\beta_i\rangle)$ is not a type-2 complexity class under the unit-cost model. By Lemma 107, there exists a total continuous H such that, for every $i \in \mathbf{N}$, $F_{\beta_i} \leq_{\mathbf{2}}^{*} H$. Since $\langle\beta_i\rangle$ is strongly convergent, it follow that H is computable. $\square$

Note that, under the answer-length-cost model, the functional $G^k_{\tau,a}$ is not necessarily in the union class $\bigcup_{c\in\mathbf{N}} \mathbf{C_A}(c\beta)$, because the amount of the cost for testing $\tau \subset f$ and the parity of $f(d)$ is no longer a fixed constant, and hence for any $c \in \mathbf{N}$, the budget received from $c\beta$ may not be sufficient for any $\mathrm{OTM_A}$ that computes $G^k_{\tau,a}$ on every (f, x) in case that f has a sufficiently large value at some point needs to be checked outside the minimal locking fragment of β.

After Corollary 132, it seems that we may have a situation in which however strong the hypotheses do not yield a nontrivial type-2 union theorem under the unit-cost model.

Conjecture 133 *There exist a total computable functional* $H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ *and a uniform, ascending, useful, and strongly convergent* $\langle \beta_i \rangle$ *such that, for every* $i \in \mathbf{N}$, $F_{\beta_i} \leq H$ *and* $\mathbf{C_U}(\langle \beta_i \rangle)$ *is not a type-2 complexity class.*

6.2 Gap and Anti-Gap Theorems

The operation of an effective operator of type $\mathcal{R} \times \mathbf{N} \to \mathbf{N}$ is indeed a special case of type-2 computation by restricting the type-1 input to $\mathcal{R}$ (recursive functions). However, the Operator Gap Theorem [Con72, You73][2] does not imply that the gap phenomenon lifts to a type-2 setting. As we did for the type-2 analog of the Union Theorem, we observe that the two different OTM models, the answer-length-cost model and the unit-cost model, have different effects on the gap phenomenon in type-2 complexity classes.

Let $e \in \mathbf{N}$ and $\beta \in \mathbf{T}_2\mathbf{TB}$. Recall that $\mathrm{OTM_A}$ stands for OTM under the answer-length-cost model, and $\widehat{\varphi}_e \in \mathbf{C_A}(\beta)$ if and only if $E_{e,\beta} = \{(f,x) | \widehat{\varphi}_{e,\beta}(f,x) \Uparrow\}$ is compact in $\mathbb{T}(\widehat{\varphi}_e)$. Similarly, we use $\mathrm{OTM_U}$, $\widehat{\varphi}^u_e$, $\mathbf{C_U}(\beta)$, and $E^u_{e,\beta}$ under the unit-cost model. Suppose that $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$ is a minimal locking fragment of β. As we have observed in previous section, if every $\mathrm{OTM_A}$ that computes $\widehat{\varphi}_e$, on input (f, a) with $\sigma \subset f$, has to query outside $\mathsf{dom}(\sigma)$, then $\widehat{\varphi}_e \notin \mathbf{C_A}(\beta)$. This is because the length of the answer from the oracle could be arbitrarily larger than $\beta(\sigma, a)$, and hence $E_{e,\beta}$ is not compact in $\mathbb{T}(\widehat{\varphi}_e)$. Thus, we can easily lift the gap phenomenon in type-1 to type-2 by ignoring every functional that queries beyond the minimal locking fragments of β, since any of such functional will never be in $\mathbf{C_A}(\beta)$, neither is in $\mathbf{C_A}(g \circ \beta)$ when $g : \mathbf{N} \to \mathbf{N}$ is computable. On the other hand, an $\mathrm{OTM_U}$ that computes a functional in $\mathbf{C_U}(\beta)$ may ask some questions outside the domain of some

[2]See Theorem 98 on page 124

minimal locking fragment of β, provided only a few fixed bits of the oracle answer are needed for the computation. Thus, we cannot ignore every oracle query when we try to create a gap in the construction of the same β. In fact, there always exists some functional with complexity between any two different type-2 time bounds, even if the two have the same locking fragments.

6.2.1 The Gap Theorem under The Answer-Length Model

We first consider the computations under the answer-length-cost model. A straightforward weak analog of the original gap theorem is that, given any recursive function $g : \mathbf{N} \to \mathbf{N}$, we want to construct a $\beta \in \mathbf{T_2TB}$ such that the gap, $\mathbf{C_A}(g \circ \beta) - \mathbf{C_A}(\beta)$, is empty. A strong analog would be the original Operator Gap Theorem where the effective operator of the theorem is restricted to the type $\mathbf{T_2TB} \to \mathbf{T_2TB}$. We will see in Section 6.3 that a strong analog of the gap phenomenon does not exist.

Since only those $\widehat{\varphi}_e$'s not in $\mathbf{C_A}(\beta)$ have to be considered when we try to keep the gap empty, we analyze the following cases of noncompact $E_{e,\beta}$. Suppose that $E_{e,\beta}$ is noncompact in $\mathbb{T}(\widehat{\varphi}_e)$.

Case 1: There exist only finitely many different $x \in \mathbf{N}$ such that, for some $\sigma \in \mathcal{F}$, $(\!(\sigma, x)\!) \subset E_{e,\beta}$, $(\!(\sigma, x)\!)$ is open in $\mathbb{T}(\widehat{\varphi}_e)$, and we have either

1. $(\!(\sigma, x)\!)$ is not open in $\mathbb{T}(F_\beta)$, or
2. $(\!(\sigma, x)\!)$ is open $\mathbb{T}(F_\beta)$.

Case 2: There are infinitely many $x \in \mathbf{N}$ such that, for some σ, $(\!(\sigma, x)\!) \subset E_{e,\beta}$.

Discussion Before we discuss the above cases, we notice that, if $(\!(\sigma, a)\!)$ is not open in the topology $\mathbb{T}(F_\beta)$, then it must be the case that (σ, a) is a locking fragment of β but not a minimal one.

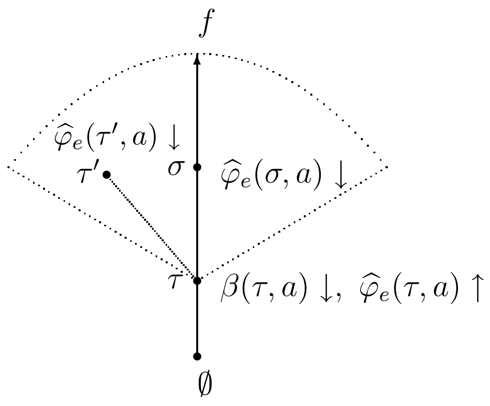

Figure 8: Non-$\mathbb{T}(F_\beta)$-Open $(\!(\sigma, a)\!)$

Case 1.1: Suppose that there is $(\!(\sigma, a)\!) \subset E_{e,\beta}$ such that, $(\!(\sigma, a)\!)$ is open in $\mathbb{T}(\widehat{\varphi}_e)$ but not open in $\mathbb{T}(F_\beta)$. Then, there must exists some $\tau \subset \sigma$ such that, (τ, a) is a locking fragment of β but not of $\widehat{\varphi}_e$ as shown in Figure 8.

Thus, any OTM that computes $\widehat{\varphi}_e$ on (f, a) with $\tau \subset f$ has to query the oracle at least one point outside $\mathsf{dom}(\tau)$. Therefore, under the answer-length-cost model, there are infinitely many $(\!(\tau', a)\!) \subset E_{e,\beta}$ such that $(\!(\tau', a)\!)$ is open in $\mathbb{T}(\widehat{\varphi}_e)$, and hence $E_{e,\beta}$ is noncompact in $\mathbb{T}(\widehat{\varphi}_e)$. Clearly, this situation will not change if we simply amplify β by some recursive function g. Because, both β and $g \circ \beta$ converge at the same point (τ, a), and $\widehat{\varphi}_e$ needs information beyond $\mathsf{dom}(\tau)$. In other words, if $\widehat{\varphi}_e \notin \mathbf{C}_{\mathbf{A}}(\beta)$ in this case, then $\widehat{\varphi}_e \notin \mathbf{C}_{\mathbf{A}}(g \circ \beta)$. Thus, we can ignore all $\widehat{\varphi}$-programs in this case in the process of constructing β for our type-2 gap theorem.

Case 1.2: Unlike Case 1.1, for some fixed a and any recursive g, having infinitely many $(\!(\sigma, a)\!) \subset E_{e,\beta}$ such that $(\!(\sigma, a)\!)$ is both $\mathbb{T}(\widehat{\varphi}_e)$-open and $\mathbb{T}(F_\beta)$-open does not imply that $\widehat{\varphi}_e \notin \mathbf{C}_{\mathbf{A}}(g \circ \beta)$. The easiest way to deal with this case is to remove it. Thus, we can simply force β to converge at $(\emptyset, x)$ for every $x \in \mathbf{N}$.

In such a way, every $\widehat{\varphi}$-program that makes any oracle query will immediately fall into Case 1.1 and be ignored.

Case 2: Suppose that there are infinitely many different x such that, for some σ, $(\!(\sigma, x)\!) \subset E_{e,\beta}$. Then, the type-0 component of $(f, x) \in E_{e,\beta}$ will eventually bigger than e. Thus, the trick in the proof of the original gap theorem can be used to keep the gap, $\mathbf{C_A}(g \circ \beta) - \mathbf{C_A}(\beta)$, empty.

Theorem 134 (Weak Type-2 Gap Theorem) *Given any strictly increasing recursive function $g : \mathbf{N} \to \mathbf{N}$, there exists $\beta \in \mathbf{T_2TB}$ such that, $\mathbf{C_A}(\beta) = \mathbf{C_A}(g \circ \beta)$.*

Proof: The argument is a simple modification of the original Gap Theorem. We only have to modify the function constructed for the original gap theorem in accordance with the criteria that any type-2 time bound has to meet. Recall from Definition 103 that, for each $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$, $\widetilde{\Phi}_e(\sigma, x)$ denotes the number of steps performed by the OTM M_e on $(\sigma^{\sim 0}, x)$ up to the completion of the computation or of the first query outside $\mathsf{dom}(\sigma)$. Consider $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ computed by the program shown in Fig 9.
Claim 1: $\beta \in \mathbf{T_2TB}$. We shall prove that β is computable, nontrivial, $\mathcal{F}$-monotone, and convergent on any input $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$.

For computability, it is clear that the condition $(floor < \widetilde{\Phi}_e(\emptyset, x) \leq ceiling)$ is recursively decidable. Moreover, since there are exactly x many $\widehat{\varphi}$-programs to be checked, it follows that there are at most x many different values of $\widetilde{\Phi}_e(\emptyset, x)$ for $1 \leq e \leq x$. By pigeonhole principle, we will eventually reach some values for $floor$ and $ceiling$ such that, the condition $(floor < \widetilde{\Phi}_e(\emptyset, x) \leq ceiling)$ fails for all $\widehat{\varphi}$-programs e such that $1 \leq e \leq x$. Therefore, the program for β will eventually need not to find another gap again and finish the for loop, and then return the value $floor$. Thus, β is computable and defined on all inputs.

Program for β

  input $\sigma : \mathcal{F}$, $x : \mathbf{N}$;

  if $x = 0$

    then return 1 and stop;

  $ceiling \leftarrow \mathsf{max}(|x| + 1, \beta(\emptyset, x - 1))$;

/* By the pigeonhole principle, a gap must exist, and hence the search for such a gap in the following go-to loop always terminates. */

/* A gap is defined by $ceiling$ and $floor$. */

 find-gap:

  $floor \leftarrow ceiling$;

  $ceiling \leftarrow g(floor)$;

/* The following loop checks if the gap is empty. */

  for $e = 1$ to x do

    if $(floor < \widetilde{\Phi}_e(\emptyset, x) \leq ceiling)$ /* The gap is not empty. */

      then GoTo find-gap: /* Try another gap. */

  end for-do;

  return $floor$;

End program

Figure 9: OTM_A Algorithm for The Weak Type-2 Gap Theorem

It is clear that β is nontrivial. Since β converges at $(\emptyset, x)$ for every x, it is also clear that β is both $\mathcal{F}$-monotone and convergent.

Claim 2: $\mathbf{C_A}(\beta) = \mathbf{C_A}(g \circ \beta)$.

$\mathbf{C_A}(\beta) \subseteq \mathbf{C_A}(g \circ \beta)$

By Theorem 67 on page 94, since $\beta \leq g \circ \beta$, it follows that $\mathbf{C_A}(\beta) \subseteq \mathbf{C_A}(g \circ \beta)$.

$\mathbf{C_A}(\beta) \supseteq \mathbf{C_A}(g \circ \beta)$

Suppose $E_{e,\beta}$ is noncompact in $\mathbb{T}(\widehat{\varphi}_e)$. Consider the cases of $E_{e,\beta}$ discussed at the beginning of this sub-section. Case 1.1 can be ignored, since in which if $E_{e,\beta}$ is noncompact in $\mathbb{T}(\widehat{\varphi}_e)$, so is $E_{e,g\circ\beta}$. Case 1.2 will never happen due to the construction of β.

In Case 2, there are infinitely many $a \in \mathbf{N}$ such that $\widehat{\varphi}_{e,\beta}(f, a) \Uparrow$ for some $f \in \mathcal{T}$. Thus, there exists $a \geq e$ such that, $(f, a) \in E_{e,\beta}$. Fix such an a. Let $\sigma \subset f$ and $\mathsf{dom}(\sigma)$ be the collection of queries ever made during the course of computation of the OTM$_\mathrm{A}$ $\widehat{M_e}$ on (f, a) before it is clipped by β. We can assume that $\sigma = \emptyset$ because in case that $\sigma \neq \emptyset$, we can use the same argument for Case 1.1 to conclude that $\widehat{\varphi}_e \notin \mathbf{C_A}(g \circ \beta)$. Thus, $\beta(\emptyset, a) < \widetilde{\Phi}_e(\emptyset, a)$. By the choice of the value for $\beta(\emptyset, a)$, $\widetilde{\Phi}_e(\emptyset, a) \leq g(\beta(\emptyset, a))$ is not allowed. Since $\beta \leq g \circ \beta$, it follows that $g(\beta(\emptyset, a)) < \widetilde{\Phi}_e(\emptyset, a)$. Thus, $(f, a) \in E_{e,g\circ\beta}$. Since there are infinitely many such a, we thus conclude that $E_{e,g\circ\beta}$ is noncompact in $\mathbb{T}(\widehat{\varphi}_e)$. ☐

In fact, the type-1 function of type $\mathbf{N} \to \mathbf{N}$ constructed in the original Gap Theorem can be seen as a special case of type-2 time bounds, which ignores the $\mathcal{F}$-typed input. As we have seen, the requirements of being a $\mathbf{T_2TB}$ can be easily maintained in the construction.

6.2.2 The Anti-Gap Theorem under The Unit-Cost Model

We have seen that the gap phenomenon in type-2 is in fact directly inherited from the type-1 computation under the answer-length-cost model. For the unit-cost model, any possible way to find an empty gap must face two difficulties discussed in the following.

Given $e \in \mathbf{N}$ and $\beta \in \mathbf{T_2TB}$, suppose that $E_{e,\beta}$ is noncompact in $\mathbb{T}(\widehat{\varphi}_e)$. Again, consider the two cases of $E_{e,\beta}$ discussed in Section 6.2.1.

- **1st difficulty:** We cannot simply force β to converge at $(\emptyset, x)$ for all $x \in \mathbf{N}$ to avoid Case 1.2 and ignore Case 1.1 as we do under the answer-length-cost model. Because, an $\mathrm{OTM_U}$ can check a portion of the type-1 argument at a fixed amount of cost. Thus, any functional that refers to any part of its type-1 input has to be considered when we try to empty the gap. The problem is that, we don't know, of course, which part of the type-1 input is needed in advance. An naive solution is to replace the condition "$floor < \widetilde{\Phi}_e(\emptyset, x) \leq ceiling$" in the algorithm of Fig 9 by "$floor < \widetilde{\Phi}_e(\sigma, x) \leq ceiling$". However, this simple change does not work. Here is a counterexample. Suppose that e is the least index of $\widehat{\varphi}$-program that computes $\widehat{\varphi}_e$, and for every $(f, x) \in E_{e,\beta}$, we have $x < e$. Then, given any $(f, x) \in E_{e,\beta}$, the algorithm will never consider the $\widehat{\varphi}$-program e in defining $\beta(\sigma, x)$ for all $\sigma \subset f$. Thus, the emptiness of the gap is not guaranteed.

- **2nd difficulty:** In order to fix the problem discussed above, $\widehat{\varphi}$-program e has to be considered infinitely often in defining $\beta(\sigma, x)$ when $x < e$. One may further replace the for-loop in the same algorithm by "for $e = 1$ to $\langle \sigma, x \rangle$ do ..." or any other alternative that guarantees any $\widehat{\varphi}$-program will be eventually considered. But this introduces another problem. Fix $(f, x) \in \mathcal{T} \times \mathbf{N}$. As the finite fragment $\sigma \subset f$ becomes larger and larger, more and more $\widehat{\varphi}$-programs will participate in defining $\beta(\sigma, x)$. Thus, β may fail to be convergent.

The two problems pointed out above cannot be solved in the same time, so we should expect to have the negation of the Weak Type-2 Gap Theorem under the unit-cost model. Our first step in that direction is the following theorem.

Theorem 135 *For every $\beta \in \mathbf{T_2TB}$, there exists a recursive function $g : \mathbf{N} \to \mathbf{N}$ such that $\mathbf{C_U}(\beta) \subset \mathbf{C_U}(g \circ \beta)$.*

Proof: Fix $\beta \in \mathbf{T_2TB}$. Consider the functional $G^k_{\tau,a}$ defined in Definition 129 on page 164, where $(\tau, a) \in \mathcal{F} \times \mathbf{N}$ is a minimum locking fragment of β and $k = \beta(\tau, a)$. We know that $G^k_{\tau,a} \notin \mathbf{C_U}(\beta)$. Let $d = |\mathsf{max}(\mathsf{dom}(\tau)) + 1|$. From the definition of $G^k_{\tau,a}$, we can have an $\mathrm{OTM_U}$ that, on every $(f, x) \in \mathcal{T} \times \mathbf{N}$, computes $G^k_{\tau,a}$ with cost

$$|x| + \|\tau\| + d + k + c,$$

for some constant c. Let $c' = \|\tau\| + d + k + c$. Since c' is a fixed constant and $\beta(\sigma, x) \geq |x| + 1$, it follows that if $g(x) \geq x + c'$ for every $x \in \mathbf{N}$, then $g \circ \beta$ can cover the expenses, and hence $G^k_{\tau,a} \in \mathbf{C}(g \circ \beta)$.

Moreover, since $\beta \leq g \circ \beta$, by Theorem 67, we have $\mathbf{C_U}(\beta) \subseteq \mathbf{C_U}(g \circ \beta)$. □

Note that the above theorem is not the negation of the Weak Gap Theorem. For an anti-gap theorem, we need a recursive g that works for every $\beta \in \mathbf{T_2TB}$.

Theorem 136 *Suppose $g : \mathbf{N} \to \mathbf{N}$ is recursive and, for every $x \in \mathbf{N}$, $g(x) \geq 3x$. Then, for every $\beta \in \mathbf{T_2TB}$, if β has a minimal locking fragment which is β-checkable, then $\mathbf{C_U}(\beta) \subset \mathbf{C_U}(g \circ \beta)$.*

Proof: Fix any $\beta \in \mathbf{T_2TB}$ with a minimal locking fragment (σ, a) that is β-checkable. Since $\beta \leq g \circ \beta$, by Theorem 67, we have $\mathbf{C_U}(\beta) \subseteq \mathbf{C_U}(g \circ \beta)$. To prove $\mathbf{C_U}(g \circ \beta) - \mathbf{C_U}(\beta) \neq \emptyset$, suppose that $\beta(\sigma, a) = k$. Since (σ, a) is β-checkable, it follows that,

$$k \geq \|\sigma\| + |a| + 1.$$

Consider the functional $G^k_{\sigma,a}$ defined in Definition 129, which is not in $\mathbf{C}_{\mathbf{U}}(\beta)$. Suppose $\mathsf{max}(\mathsf{dom}(\sigma)) + 1 = d$. We can construct an $\mathrm{OTM}_{\mathrm{U}}$, $\widehat{M}^u_e$, that computes $G^k_{\sigma,a}$ with complexity bounded as described below. For every $(f, x) \in \mathcal{T} \times \mathbf{N}$, there are two cases:

Case 1: $\sigma \not\subset f$ or $x \neq a$. Since (σ, a) is β-checkable, it is legitimate to assume that the β-clocked machine, $\widehat{M}^u_{e,\beta}$, can check this case and output 0 under the budget provided by β. Thus, so can $\widehat{M}^u_{e,g\circ\beta}$.

Case 2: $\sigma \subset f$ and $x = a$. The machine, $\widehat{M}^u_e$, after checking $\sigma \subset f$ and $x = a$, needs to take some further actions, i.e., output 0 or 2^{k+1} in accordance with the parity of $f(d)$. Under the unit-cost model, these actions cost $|d| + k + 2$ steps. In other words, the complexity of computing $\widehat{M}^u_e(f, a)$ is bounded as follows:

$$\widehat{\Phi}^u_e(f, a) \leq (\|\sigma\| + |a| + 1) + |d| + k + 2.$$

According to our conventions on coding, we have

$$|d| = |\mathsf{max}(\mathsf{dom}(\sigma)) + 1| \leq \|\sigma\| + 1.$$

Since (σ, a) is β-checkable, it follows that,

$$\|\sigma\| + |a| + 1 \leq \beta(\sigma, a) = k.$$

Thus,

$$\widehat{\Phi}^u_e(f, a) \leq (\|\sigma\| + |a| + 1) + |d| + k + 2 \leq 3k = g(\beta(\sigma, a)).$$

This means that, $\widehat{M}_{e,g\circ\beta}$, on (f, x) in this case, computes $G^k_{\tau,a}(f, x)$.

Therefore, $\widehat{M}^u_{e,g\circ\beta}$ computes $G^k_{\tau,a}$ in all cases, and hence $G^k_{\tau,a} \in \mathbf{C}(g \circ \beta)$. $\square$

Recall from Definition 56 that a $\beta \in \mathbf{T_2TB}$ is called *useful* iff for every minimal locking fragment of β, (σ, x), we have

$$\forall \tau \subseteq \sigma[\beta(\tau, x) \geq \|\tau\| + |x| + 1].$$

Corollary 137 *Suppose $g : \mathbf{N} \to \mathbf{N}$ is recursive and, for every $x \in \mathbf{N}$, $g(x) \geq 3x$. Then, for every useful $\beta \in \mathbf{T_2TB}$, $\mathbf{C_U}(\beta) \subset \mathbf{C_U}(g \circ \beta)$.*

Proof: This corollary follows directly from Definition 56 and Theorem 136. □

Theorem 138 (Anti-Gap Theorem) *Suppose $g : \mathbf{N} \to \mathbf{N}$ is recursive and, for every $x \in \mathbf{N}$, $g(x) \geq 3x$. Then, for every strong $\mathcal{F}$-monotone $\beta \in \mathbf{T_2TB}$, $\mathbf{C_U}(\beta) \subset \mathbf{C_U}(g \circ \beta)$.*

Proof: We shall prove that the function g in Theorem 136 also works for $\beta \in \mathbf{T_2TB}$ in case that no locking fragment of β is β-checkable provided that β is strong $\mathcal{F}$-monotone.

Suppose $\beta \in \mathbf{T_2TB}$ such that, every locking fragment of β is not β-checkable. Fix a $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$ such that, (σ, a) is β-checkable but all extensions of σ are not β-checkable. Such (σ, a) must exist, otherwise, by the assumption that all locking fragments of β are not β-checkable and König's lemma, there exists $f \in \mathcal{T}$ such that, F_β does not converge on (f, a).

Case 1: $\sigma = \emptyset$. Consider the following functional, $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$,

$$F(f, x) = \begin{cases} 1 & \text{if } x = a \text{ and } f(0) \text{ is odd;} \\ 0 & \text{otherwise.} \end{cases}$$

On input (f, a) where $f \in \mathcal{T}$ and $f(0)$ is odd, the minimum cost of any OTM_U that computes F is a constant dominated by the size of a. Let the cost be k. Clearly, k is the same as the cost of checking $(\{(0,1)\}, a)$, because only the least significant bit of the value of $f(0)$ has to be scanned. Thus,

$$k = |a| + 1 + \|\{(0,1)\}\| = |a| + 5.$$

But, $(\{(0,1)\}, a)$ is not β-checkable, so we have

$$|a| + 5 > \beta(\{(0,1)\}, a) \geq |a| + 1.$$

In other words, the budget provided by β is not enough for any OTM_U, M_e, that computes F to scan the first bit of the value of $f(0)$ in case that $f(0)$ is odd. Thus, $\widehat{\varphi}_{e,\beta}(f, a) \Uparrow$. A similar argument can be used to the case that $f(0)$ is even to conclude that $\widehat{\varphi}_{e,\beta}(f, a) \Uparrow$. Therefore, for each $y \in \mathbf{N}$, $(\!(\{(0,y)\}, a)\!) \subset E_{e,\beta}$ and, moreover, $(\!(\{(0,y)\}, a)\!)$ is an open set in $\mathbb{T}(\widehat{\varphi}_e)$. It follows that $E_{e,\beta}$ is noncompact in $\mathbb{T}(\widehat{\varphi}_e)$ and hence $F \notin \mathbf{C}_\mathbf{U}(\beta)$.

Clearly, if $|a| \geq 2$, then

$$g(\beta(\{(0,1)\}, a)) \geq 3(|a| + 1) > |a| + 5.$$

Thus, we can have an OTM_U clocked by $g \circ \beta$ that computes F. Hence $F \in \mathbf{C}_\mathbf{U}(g \circ \beta)$.

Case 2: $\sigma \neq \emptyset$. Let $n \in \mathbf{N}$ be the least number not in $\mathsf{dom}(\sigma)$. Consider the following functional, $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$,

$$F(f, x) = \begin{cases} 1 & \text{if } x = a, \sigma \subset f, \text{ and } f(n) \text{ is odd;} \\ 0 & \text{otherwise.} \end{cases}$$

On input (f, a) where $f \in \mathcal{T}, \sigma \subset f$, and $f(n)$ is odd, the minimum cost of any OTM_U that computes F is a constant, k, dominated by $|a|$ and $\|\sigma\|$, and k is the same as the cost of checking $(\sigma \cup \{(n,1)\}, a)$, which is

$$k = \|\sigma \cup \{(n,1)\}\| + |a| + 1 = \|\sigma\| + |a| + |n| + 3.$$

By the assumption, (σ, a) is β-checkable and all its extensions are not, and since β is strong $\mathcal{F}$-monotone,

$$\|\sigma\| + |a| + |n| + 3 > \beta(\sigma \cup \{(n,1)\}, a) \geq \beta(\sigma, a) \geq \|\sigma\| + |a| + 1.$$

Since n is the least number not in $\mathsf{dom}(\sigma)$, it follows that $|n| \leq \|\sigma\| + 1$. Also, since $|a| \geq 1$, we have

$$g(\beta(\sigma \cup \{(n,1)\}, a)) \geq g(\beta(\sigma, a)) \geq 3(\|\sigma\| + |a| + 1) \geq \|\sigma\| + |a| + |n| + 3.$$

Similarly, when $f(n)$ is even. For $(f, x) \in \mathcal{T} \times \mathbf{N}$ and $\sigma \not\subset f$ or $x \neq a$, since (σ, a) is β-checkable, $\widehat{M}_e^u$ clocked by β, on such (f, x), can finish the checking and output 0. Thus, $F \in \mathbf{C_U}(g \circ \beta)$.

What remains to argue is that $F \notin \mathbf{C_U}(\beta)$. Fix any $\mathrm{OTM_U}$, $\widehat{M}_e^u$, that computes F. Suppose that q is the last query asked among $\mathsf{dom}(\sigma) \cup \{n_0\}$ throughout the course of computation of $\widehat{M}_e^u$ on (f, a) where $\sigma \subset f$. If $q = n$, we can slightly modify the same argument in the previous case to conclude that $F \notin \mathbf{C_U}(\beta)$. If $q \in \mathsf{dom}(\sigma)$, consider τ_y for each $y \in \mathbf{N}$ defined in the following.

$$\tau_y = (\sigma - \{(q, \sigma(q))\}) \cup \{(q, \sigma(q) + y \cdot 2^{|\sigma(q)|}), (n, 1)\}.$$

Thus, for every $y \in \mathbf{N}$ and $f \in \mathcal{T}$, if $\tau_y \subset f$, then $\widehat{\varphi}_{e,\beta}(f, a) \Uparrow$. This is because $(\sigma \cup \{(n,1)\}, a)$ is not β-checkable, and after the first $|\sigma(q)|$ bits of the value of $f(q)$ scanned, the budget for $\widehat{M}_e^u$ must be used up. Thus, for every $y \in \mathbf{N}$, $(\!(\tau_y, a)\!) \subset E_{e,\beta}$ and is an open set in $\mathbb{T}(\widehat{\varphi}_e)$. Therefore, $F \notin \mathbf{C_U}(\beta)$. $\square$

Note that we can disprove Theorem 138 if we remove the strong $\mathcal{F}$-monotonicity restriction of β. The idea is that, β shrinks the budget to the bottom, $|x| + 1$, until it receives a locking fragment that is too long to be seen under the budget $g \circ \beta$ can provide.

6.3 Compression Theorems

The previous section shows that if the gap is defined by a type-1 recursive function, then the existence of the weak gap phenomenon depends on the cost model of OTMs. We now consider a stronger form of the gap phenomenon where the gap is defined by an effective operator. The type of the effective operator Θ we interested is that $\Theta : \mathbf{T_2TB} \to \mathbf{T_2TB}$. Thus, the analogous operator gap theorem on $\mathbf{T_2TB}$ does not immediately follow the original Operator Gap Theorem described in Section 5.3. In fact, the gap does not exist among those type-2 complexity classes under either OTM cost model if we restrict the effective operator to type $\mathbf{T_2TB} \to \mathbf{T_2TB}$.

In type-1, the theorem that opposes the gap phenomenon is called *compression theorem* which spells out the fact that, under the condition that the source of time bounds is restricted to a fixed honest set, we can compress a computable function with complexity between any two such time bounds.[3] We follow this convention and call our theorem that describes a weak form of the anti-gap phenomenon *weak compression theorem*. Our strong type-2 analog of the Compression Theorem, we call the *Inflation Theorem* to reflect the fact that we can uniformly inflate any type-2 complexity class via certain effective operators. We will see in a moment that, unlike the type-1 Compression Theorem, the restriction of time bounds to a fixed honest set is not needed in type-2.

Recall that when the choice of the cost model for our OTMs has no effect on our theorems, we simply use $\mathbf{C}(\beta)$ to denote the type-2 complexity class instead of $\mathbf{C_A}(\beta)$ and $\mathbf{C_U}(\beta)$ for the two models, respectively.

[3] See Section 5.3 for more details.

6.3.1 Weak Compression

We first note a weaker theorem that can be obtained directly from Theorem 82. The following theorem asserts that, given an index of any type-2 time bound β, we can uniformly construct a computable type-2 functional F such that, F is not in the complexity class $\mathbf{C}(\beta)$.

Theorem 139 (Weak Compression Theorem) *There is a recursive function r such that, for every $i \in \mathbf{N}$, if $\varphi_i \in \mathbf{T_2TB}$, then $\widehat{\varphi}_{r(i)} \notin \mathbf{C}(\varphi_i)$.*

Proof: Recall Theorem 82 and its proof. In the construction we clock every $\widehat{\varphi}$-program e on some inputs to see if e needs to be canceled. We can select one of the two clocking schemes in accordance with the OTM cost model under consideration. Thus, the construction can be applied to both cost models. Also, note that we can uniformly construct the functional F in Theorem 82. Thus, with an appropriate version of the S-m-n Theorem discussed in Section 1.5, we can extend Theorem 82 to our Weak Compression Theorem immediately. $\square$

6.3.2 Strong Compression

In this subsection we show that the gap phenomenon does not occur under the power of effective operators on type-2 time bounds, and the cost model for OTMś is not important. We shall construct an effective operator, $\Theta : \mathbf{T_2TB} \to \mathbf{T_2TB}$, such that, for any $\beta \in \mathbf{T_2TB}$, we have: $\Theta(\beta) \in \mathbf{T_2TB}$ and

$$\mathbf{C}(\Theta(\beta)) \subset \mathbf{C}(\Theta^2(\beta)) \subset \mathbf{C}(\Theta^3(\beta)) \subset \cdots .$$

Thus, with such Θ, we can have a strictly increasing hierarchy of type-2 complexity classes from any given β. In other words, we can uniformly inflate any type-2 complexity class forever.

The idea of the construction of such Θ is quite easy. Given a $\beta \in \mathbf{T_2TB}$, we know that by definition, it must be convergent. Clearly, there always exist some functionals that try to query the oracle at some non-β-queriable points and simply output the oracle answers. Any machine, under either cost model, that computes such functionals will be clipped by β at the moment the machine attempts to query the oracle at a non-β-queriable point. The task of our Θ is to calculate a non-β-queriable point and output enough budget to let some machines be able to finish the oracle query and print out the oracle answer afterwards. Since the functionals we want to include in the inflated complexity class are those that print out the entire oracle answer, the difference between two cost models becomes not important. Note that the β-checkability is undecidable under the answer-length-cost model. This is the reason why we construct Θ to find a non-β-queriable point instead of a non-β-checkable one. The following theorem is shown by an easy construction of a such Θ that can inflate any given type-2 complexity class.

Theorem 140 (Inflation Theorem) *There is a recursive operator,* $\Theta : \mathbf{T_2TB} \to \mathbf{T_2TB}$ *such that, for each* $\beta \in \mathbf{T_2TB}$*, we have* $\Theta(\beta) \in \mathbf{T_2TB}$ *and* $\mathbf{C}(\beta) \subset \mathbf{C}(\Theta(\beta))$.

Proof: We construct a $\Theta : \mathbf{T_2TB} \to \mathbf{T_2TB}$ as follows. Θ takes a $\beta \in \mathbf{T_2TB}$ as its input and outputs the program, $\Theta(\beta)$, shown in Figure 10.

It is clear that Θ is a recursive operator. Suppose $\beta \in \mathbf{T_2TB}$. We shall prove the following two claims.

Claim 1: $\Theta(\beta) \in \mathbf{T_2TB}$**.**

$\Theta(\beta)$ is computable, nontrivial, $\mathcal{F}$-monotone, and convergent. Thus, $\Theta(\beta) \in \mathbf{T_2TB}$. Note that, if β is a strong $\mathcal{F}$-monotone type-2 time bound, then so is $\Theta(\beta)$.

```
Program for Θ(β)
    input σ : F; x : N;
    k ⟵ β(σ, x) + 1;
    τ ⟵ { ⟨i, σ(i)⟩ | i ∈ dom(σ) ∧ i ≤ 2^k };
    output k + 2 · ‖τ‖;
End program
```

Figure 10: Program for $\Theta(\beta)$ in The Inflation Theorem

Claim 2: $\mathbf{C}(\beta) \subset \mathbf{C}(\Theta(\beta))$.

According to the construction for $\Theta(\beta)$, we know that $\beta \leq \Theta(\beta)$, and hence $\mathbf{C}(\beta) \subseteq \mathbf{C}(\Theta(\beta))$.

For $\mathbf{C}(\beta) \neq \mathbf{C}(\Theta(\beta))$, consider the function $f_0 = \lambda x : \mathbf{N}.0$. Fix any $a \in \mathbf{N}$. Let (σ_0, a) with $\sigma_0 \subset f_0$ be a minimal locking fragment of β. Suppose that $\beta(\sigma_0, a) = k$, and let $\tau_0 = f_{0[2^k+1]}$. Thus, τ_0 is an initial segment of f_0 with $\mathsf{max}(\mathsf{dom}(\tau_0)) = 2^k$. Recall that τ_0^- is the initial segment of τ_0 which is one element shorter than τ_0. Consider the following functional $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$,

$$F(f, x) = \begin{cases} 1 & \text{if } x = a, \tau_0^- \subset f, \text{ and } f(2^k) \text{ is odd;} \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, (τ_0, a) is a minimal locking fragment of F, and it is also clear that $F \notin \mathbf{C}(\beta)$. For $F \in \mathbf{C}(\Theta(\beta))$, we can construct a $\widehat{\varphi}$-program e such that, on any input $(f, x) \in \mathcal{T} \times \mathbf{N}$, e checks the following in the given order:

(i) Is $x = a$?

(ii) From $n = 0$ to $n = 2^k - 1$, is $f(n) = 0$?

(iii) Is $f(2^k)$ odd?

Once a negative answer to any of the above questions turns out, e outputs 0 and halts; otherwise, outputs 1 and halts. Clearly, such $\widehat{\varphi}_e$ computes F. To see $F \in \mathbf{C}(\Theta(\beta))$, fix any $(f, x) \in \mathcal{T} \times \mathbf{N}$. We argue that $F(f, x) = \widehat{\varphi}_{e,\Theta(\beta)}(f, x)$ as follows.

Case 1: $x \neq a$.

By convention, $\|\emptyset\| = 2$ (see page 64 for details), we have

$$\Theta(\beta)(\emptyset, x) = \beta(\emptyset, x) + 1 + 2 \cdot \|\emptyset\| \geq |x| + 6.$$

Since $x \neq a$ can be detected in $|x| + 1$ steps and two more steps are needed to output 0, the budget $|x| + 6$ is enough for this computation. Therefore, in this case $\widehat{\varphi}_{e,\Theta(\beta)}(f, x) \Downarrow = 0$.

Case 2: $x = a$.

We prove by induction that, for every $\sigma \in \mathcal{F}$ such that σ is some initial segment of f, $\widetilde{\Phi}_e(\sigma, x) \leq \Theta(\beta)(\sigma, x)$. This is sufficient to prove that $F(f, x) = \widehat{\varphi}_{e,\Theta(\beta)}(f, x) \Downarrow$ in this case.

• For inductive basis, assume $\sigma = \emptyset$. Note that the arguments in Case 1 are not quite sufficient to prove $\widetilde{\Phi}_e(\emptyset, x) \leq \Theta(\beta)(\emptyset, x)$, because the cost of querying f at 0, which costs two steps, is included in $\widetilde{\Phi}_e(\emptyset, x)$. On the other hand, the OTM does not output 0 at this moment. This saves exactly the same cost for the query 0. Thus, the inductive basis is proved.

• For inductive hypothesis, suppose $\sigma \in \mathcal{F}$ with $\sigma \subset f$ and, for every $\tau \subset \sigma$, $\widetilde{\Phi}_e(\tau, x) \leq \Theta(\beta)(\tau, x)$.

• For inductive step, we shall prove that $\widetilde{\Phi}_e(\sigma, x) \leq \Theta(\beta)(\sigma, x)$. Clearly, if $\tau_0 \subset \sigma$, then by the hypothesis and that τ_0 is a minimal locking fragment of both $\widehat{\varphi}_e$ and $\Theta(\beta)$, it follows that $\widetilde{\Phi}_e(\sigma, x) \leq \Theta(\beta)(\sigma, x)$.

Suppose $\sigma \subseteq \tau_0$. If there is some $\tau \subset \sigma$ such that (τ, x) is a minimal locking fragment of $\widehat{\varphi}_e$, then

$$\widetilde{\Phi}_e(\sigma, x) = \widetilde{\Phi}_e(\tau, x) \leq (x+1) + \|\tau\| + 2 \leq (x+1) + \|\sigma\| + 2 \leq \Theta(\beta)(\sigma, x).$$

One the other hand, if, for every $\tau \subset \sigma$, (τ, x) is not a locking fragment of $\widehat{\varphi}_e$, then we consider the computation of $\widehat{M_e}$ on (f, x). Clearly, the first oracle query, q, outside $\mathsf{dom}(\sigma^-)$ is $\mathsf{max}(\mathsf{dom}(\sigma))$. After query q made, under the answer-length-model, the budget will be updated with an extra amount, $|q|+|\sigma(q)|+2$, immediately. This allows the computation to go on and scan $\sigma(q)$ and make next query $q+1$. Thus, $\widetilde{\Phi}_e(\sigma, x) \leq \Theta(\beta)(\sigma, x)$. For the unit-cost model, consider the remaining budget after the query q is made, which is

$$\left[\beta(\sigma^-, x) + 1 + 2 \cdot \|\sigma^-\|\right] - \left[(|x|+1) + \|\sigma^-\| + (|q|+1)\right]. \tag{45}$$

Since $\beta(\sigma^-, x) \geq |x| + 1$ and $\|\sigma^-\| \geq |q| + 2$, it follows that (45) ≥ 2. This allows the $\mathrm{OTM_U}$ to scan two bits from the oracle-answer tape and check if $f(q) = 0$. Whether $f(q) = 0$ or not, after the first two bits of the oracle answer scanned, the budget will increase at least by $|q| + 3$. Thus, the $\mathrm{OTM_U}$ has enough remaining budget to output 0 or make next query $q + 1$. Therefore, $\widetilde{\Phi}_e(\sigma, x) \leq \Theta(\beta)(\sigma, x)$.

Therefore, in all cases, $\widehat{\varphi}_{e,\Theta(\beta)}(f, x) = F(f, x)$, and hence $F \in \mathbf{C}(\Theta(\beta))$. $\square$

Recall from Definition 104 that a uniform $\langle \beta_i \rangle$ is a sequence of type-2 time bounds such that, $\lambda i, \sigma, x.\beta_i(\sigma, x)$ is recursive. Also, recall that $\mathbf{C}(\langle \beta_i \rangle)$ is the union class, $\bigcup_{i \in \mathbf{N}} \mathbf{C}(\beta_i)$, and, in general, it is not a type-2 complexity class. Let Θ be the operator above and $\beta \in \mathbf{T_2TB}$.

Let $\beta_i = \Theta^i(\beta)$. According to the construction of Θ, it is clear that $\langle \beta_i \rangle$ is not convergent, and hence the union class $\mathbf{C}(\langle \beta_o \rangle)$ is not a type-2 complexity class.

Moreover, Section 6.1 showed that, even if $\langle\beta_i\rangle$ is uniform, ascending, and uniformly convergent, $\mathbf{C}(\langle\beta_i\rangle)$ is not necessarily a type-2 complexity class. Thus, Theorem 140 does not immediately give a corollary that we can uniformly inflate any union class, $\mathbf{C}(\langle\beta_i\rangle)$, by simply applying the recursive operator constructed in the inflation theorem to the type-2 time bound that determines $\mathbf{C}(\langle\beta_i\rangle)$. Moreover, even if we apply the effective operator constructed for Theorem 140 to each time-bound in $\langle\beta_i\rangle$, the union class $\mathbf{C}(\langle\Theta(\beta_i)\rangle)$ is not necessarily to properly include the union class $\mathbf{C}(\langle\beta_i\rangle)$. We construct such a $\langle\beta_i\rangle$ in the next theorem.

Theorem 141 *For every recursive operator* $\Theta : \mathbf{T}_2\mathbf{TB} \to \mathbf{T}_2\mathbf{TB}$ *such that, for every* $\beta \in \mathbf{T}_2\mathbf{TB}$, $\mathbf{C}(\beta) \subset \mathbf{C}(\Theta(\beta))$, *there is a uniform ascending* $\langle\beta_i\rangle$ *such that* $\mathbf{C}(\langle\beta_i\rangle) = \mathbf{C}(\langle\Theta(\beta_i)\rangle)$.

Proof: : Fix any type-2 time bound β. Define $\beta_0 = \beta$ and $\beta_i = \Theta(\beta_{i-1})$ for $i \geq 1$. Clearly, if Θ is a recursive operator and $\beta \in \mathbf{T}_2\mathbf{TB}$, then $\langle\beta_i\rangle$ is uniform. Moreover, since, for all $i \in \mathbf{N}$, $\beta_i \leq \Theta(\beta_i)$, $\langle\beta_i\rangle$ is ascending. Since $\mathbf{C}(\beta_i) \subseteq \mathbf{C}(\beta_{i+1})$, it follows that

$$\mathbf{C}(\langle\beta\rangle) = \bigcup_{i\geq 0}\mathbf{C}(\beta_i) = \bigcup_{i\geq 1}\mathbf{C}(\beta_i) = \bigcup_{i\geq 0}\mathbf{C}(\Theta(\beta_i)) = \mathbf{C}(\langle\Theta(\beta_i)\rangle).$$

□

The next question then is that: What can we do on a given uniform $\langle\beta_i\rangle$? Can we inflate the union class determined by a $\langle\beta_i\rangle$, especially when $\mathbf{C}(\langle\beta_i\rangle)$ is not a type-2 complexity class? According to Theorem 141, the answer to the uniform version of the question is simply no. Nevertheless, we have the following theorem asserting that, given any uniform ascending $\langle\beta_i\rangle$, we can construct a recursive operator to inflate $\mathbf{C}(\langle\beta_i\rangle)$. However, the construction is not uniform, i.e., the construction depends on the given $\langle\beta_i\rangle$.

Theorem 142 *Given any uniform ascending $\langle\beta_i\rangle$, there exists a recursive operator $\Theta : \mathbf{T_2TB} \to \mathbf{T_2TB}$ such that, $\mathbf{C}(\langle\beta_i\rangle) \subset \mathbf{C}(\langle\Theta(\beta_i)\rangle)$.*

Proof: : By Theorem 112, given any uniform ascending $\langle\beta_i\rangle$, there exists a total computable functional $\widehat{\varphi}_p$ such that $\widehat{\varphi}_p \notin \mathbf{C}(\langle\beta_i\rangle)$. Fix such $\widehat{\varphi}$-program e, and define $\alpha : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ as follows. For every $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, let

$$\alpha(\sigma, x) = \mathsf{max}(\{\widehat{\Phi}_p(\tau^{0\sim}, x) \mid \tau \subset \sigma\}).$$

Clearly, $\alpha \in \mathbf{T_2TB}$ and $\widehat{\varphi}_p \in \mathbf{C}(\alpha)$. Also, define $\Theta : \mathbf{T_2TB} \to \mathbf{T_2TB}$, for every $\beta \in \mathbf{T_2TB}$ and $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$,

$$\Theta(\beta)(\sigma, x) = \mathsf{max}(\alpha(\sigma, x), \beta(\sigma, x)).$$

Thus, for every β_i, $\widehat{\varphi}_p \in \mathbf{C}(\Theta(\beta_i))$, and hence $\widehat{\varphi}_p \in \mathbf{C}(\langle\Theta(\beta_i)\rangle)$. □

6.4 Honesty in Type-2 Computation

One of the key notions in complexity theory is that of honesty. We say that a function f is *honest* in a sense that, for all but finitely many x, we do not need too much resource in order to compute a small value $f(x)$. Formally,

Definition 143 *Suppose $f, g : \mathbf{N} \to \mathbf{N}$. We say that f is g-honest if and only if there is an $e \in \mathbf{N}$ such that, $\Phi_e \leq^* g \circ f$.* ■

Clearly, there are many very *dishonest* function(al)s, e.g., the 0-1 valued function in Rabin's theorem [Rab60] and the 0-1 valued functional in our Theorem 82 (see Section 4.2.4).

We also want to know if any complexity class can be determined by some honest function. McCreight and Meyer [MM69] point out that every Blum's measured set is a subset of some collection of g-honest functions. However, not every g-honest function is in the measured set. Thus, the question becomes: Is it necessarily to lose some complexity classes if we restrict our complexity-bound functions to a measured set? McCreight and Meyer's [MM69] Honesty Theorem (see Theorem 99) answers this question and asserts that we can construct a measured set, $\mathcal{M}$, such that every complexity class can be determined by one of the functions in $\mathcal{M}$. In other words, every complexity class has an honest name. Since our type-2 time bounds are type-1 functions, we can simply follow Blum's original definitions for measured sets and, hopefully, we can construction a measured set, $\mathcal{M}$, such that, there is at least one honest name for every type-2 complexity class in $\mathcal{M}$. Indeed we can have a type-2 analog of the Honesty Theorem as follows:

Theorem 144 *There exists a recursive function g such that $\{\alpha_{g(i)} | i \in \mathbf{N}\}$ is a measured set, and for all $\beta \in \mathbf{T}_2\mathbf{TB}$, if $\beta = \varphi_i$, then $\mathbf{C}(\beta) = \mathbf{C}(\alpha_{g(i)})$.*

Proof: We skip the proof because it can be obtained by a straightforward translation of the original proof of the Honesty Theorem in our $\widehat{\varphi}$-programming system. □

However, we think that this type-2 analog is not so important as the original Honesty Theorem for the following reason. With the gap phenomenon as in the Operator Gap Theorem, we need to have a measured set in order to apply the Compression Theorem, so that the gap phenomenon can be removed. And, by using the fiendish construction in the proof of Honesty Theorem, we can do so without losing any complexity class. In type-2, on the other hand, to embed the entire set of type-2 time bounds in a measured set is not necessary because of the Inflation Theorem (Theorem 140) which asserts that the gap phenomenon does not exist under the power of effective operators of type $\mathbf{T}_2\mathbf{TB} \rightarrow \mathbf{T}_2\mathbf{TB}$.

We note that the convergence property of each $\beta \in \mathbf{T}_2\mathbf{TB}$ is of help to have the positive result we just mentioned (i.e., we are free of the gap phenomenon in our clocked type-2 computation). With the convergence property, the proof of the Inflation Theorem is rather straightforward. This suggests that the trick used in the original Operator Gap Theorem is hidden behind some locking fragments, and consequently, the trick does not affect any type-2 time bound in determining a complexity class. Moreover, Theorem 88 asserts that, from any $\beta \in \mathbf{T}_2\mathbf{TB}$, we can uniformly obtain an *equivalent* and *locking detectable* type-2 time bound. It is clear that, intuitively, if we can uniformly obtain an *equivalent* and *locking detectable* type-2 time bound, then we can uniformly construct another bigger type-2 time bound that determine a bigger type-2 complexity class. We thus consider *locking detectability* as a weak version of honesty property for our type-2 time bounds. Also, we consider Theorem 88 as our type-2 analog of the Honesty Theorem.

Our last concern is *constructability*, more precisely, *time-constructability* as we just concern the number of steps an OTM performs as our complexity measure. Recall that, in type-1, a recursive function t is said to be time-constructable if and only if there is a φ-program i for t such that, for every $x \in \mathbf{N}$, we have $\Phi_i(x) = |t(x)|$. We shall keep in mind that our type-2 time bounds are in fact type-1 functions and that we require all type-0 inputs to be scanned at least once. In addition, since every $\beta \in \mathbf{T}_2\mathbf{TB}$ must be convergent, it is clear that every β is not time-constructable in the above sense. Thus, the original notion of time-constructability is too strong a property for our type-2 time bounds. We consider the following alternative.

Definition 145 (Type-2 Time-Constructable) *Let $\beta \in \mathbf{T}_2\mathbf{TB}$. We say that β is type-2 time-constructable if and only if β is useful and there is a φ-program i for β such that, for every $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, $\varphi_i(\langle \sigma, x \rangle) = \beta(\sigma, x)$ and, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$,* *there is a* $y \in \mathbf{N}$ *such that*

$$\lim_{\sigma \to f} \Phi_i(\langle \sigma, x \rangle) = y + \|\sigma\| + |x|.$$

■

In other words, a $\beta \in \mathbf{T_2TB}$ is type-2 time-constructable if and only if β is useful and, except the cost for scanning its inputs, the cost of the computation is convergent in above sense. Clearly, if β is type-2 time-constructable, then it is locking detectable and polynomial time computable.

Chapter 7

Optimal $\widehat{\varphi}$-programs

In the previous chapter we studied a naive type-2 analog of the Gap Theorem. The theorem can be obtained by directly lifting the original proof of the Gap Theorem due to the fact that any type-1 computation is just a special case of type-2 computations. We can make use of this fact again to prove a type-2 speed-up theorem. As with the gap phenomenon, the speed-up phenomenon in type-2 is inherited from type-1 computations. However, oracle queries do introduce some difficulties when we attempting a direct translation of the speed-up argument. In the first half of this chapter, we discuss the difficulties and show how to overcome them and translate the original proof of the Speed-up Theorem for our $\widehat{\varphi}$-programing system.

The second half of this chapter explores an alternative complexity measure, query complexity, that has an associated speed-up phenomenon. That is, we consider the number of queries made during the course of the computation a computational resource to be sped-up. However, the query complexity does not have the two properties described in Blum's two axioms. Thus, the original technique fails to prove a query-complexity speed-up theorem. Nevertheless, we give three notions of *query-optimal* programs and prove that there is a computable type-2 functional that has no *query-optimal* $\widehat{\varphi}$-program.

7.1 Speed-up Theorems

We say that a best program for a computable function f is a program that computes f with a cost that no other program for f can do it better on all but finitely many inputs. For obvious reasons, we want to improve our programs as much as possible and, in the best case, we can obtain a best program. Based on Blum's two axioms reviewed in Section 5.1, the Speed-up Theorem [Blu67] asserts that there is not always a best program for every computable function. When we extend the complexity theory into type-2 computations with the new power of oracle queries in our $\widehat{\varphi}$-programming system, we face the same question: Does a best $\widehat{\varphi}$-program always exist for every computable functional of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$.

Since type-1 computations are included in type-2 computations and the speedable function constructed for the original Speedup Theorem is a pure type-1 function that has nothing to do with the oracle queries, it would seem to intuitively follow that the speed-up phenomenon must be hereditary as long as the complexity measure concerned satisfies Blum's axioms. (Clearly, our standard complexity measure $\langle \widehat{\Phi}_i \rangle$, the number of steps the OTM performs, satisfies Blum's axioms.) In this section we consider the standard complexity measure, $\langle \widehat{\Phi}_i \rangle$, and construct a type-2 functional that has no best $\widehat{\varphi}$-program. Although the construction is essentially copied from Blum's the original proof of the Speed-up Theorem, there are a few difficulties in translating the construction.

7.1.1 Unnecessary but Helpful Queries

The first thing we consider is whether there is a direct lift of the speed-up phenomenon from type-1 to type-2. An intuitive approach is to code a type-1 speedable function g constructed for the original Speed-up Theorem and define $G : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that, $G(f, x) = g(x)$. However, this approach faces the following difficulty for a type-2 speed-up theorem.

Recall that the original construction of the type-1 speedable function is based on the cancellation of some pure type-1 programs when the run time of these programs fall into a certain unwanted range. When we directly life the construction to type-2, we note that there are cases where unnecessary queries may slow down or speedup the computation and that some programs may escape being canceled. Note that the proofs of the Union Theorem and Gap Theorem do not involve cancellation but construct time bounds. Thus, we did not face the current problem in lifting those constructions in the previous chapter.

To deal with this problem, we first take a closer look at the definitions of induced topologies, $\mathbb{T}_{min}(\cdot)$ (Definition 30), and the almost everywhere less relation, $\leq_{\mathbf{2}}^{*}$ (Definition 35). We begin by formally defining an *unnecessary query.*

Definition 146 (Unnecessary Queries) *Let e be a $\widehat{\varphi}$-program and $(f, x) \in \mathcal{T} \times \mathbf{N}$. Let (σ, x) with $\sigma \subset f$ be a minimal locking fragment of $\widehat{\varphi}_e$. Suppose that $Q \subset \mathbf{N}$ is the collection of the oracle queries made during the course of the computation of the OTM, $\widehat{M}_e$, on inputs f and x. We say that the queries in $Q - \mathsf{dom}(\sigma)$ are **unnecessary queries** of the computation of $\widehat{M}_e$ on input (f, x).* ■

Clearly, we can intentionally write a $\widehat{\varphi}$-program that queries arbitrarily many unnecessary queries without changing the final result of the computation. On the other hand, given a computable type-2 functional, we cannot in general write a $\widehat{\varphi}$-program that makes only necessary queries on all inputs.

Theorem 147 *There is a computable functional $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that, every $\widehat{\varphi}$-program computing F makes some unnecessary queries on some inputs.*

Proof: Define $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ by,

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

Clearly, F is computable and total. Suppose, by contradiction, that there is a $\widehat{\varphi}$-program e that computes F without making unnecessary queries on any inputs. Fix any a such that, $\varphi_a(a)\uparrow$. Then, on input (f,a), $\widehat{M_e}$ cannot query $f(0)$, because for every $g \in \mathcal{T}$, $F(g,a) = 0$, and hence $(\emptyset, a)$ is the minimal locking fragment of F on (g,a). Thus, 0 is an unnecessary query. That means e can decide the halting problem, which is impossible. Hence, F is as required. $\square$

In fact, there are computable functionals that make unnecessary queries on all inputs, and the number of unnecessary queries can be arbitrarily large. We will see more details about the query-complexity in Section 7.3.

While unnecessary queries do not, in some sense, contribute to the value of the final result, they can affect the speed at which this final result obtained. As the functional we just constructed in the proof of Theorem 147, when $\varphi_a(a)\uparrow$, the smaller the value of $f(0)$, the sooner the computation of $\widehat{M_e}$ on (f,a) halts. These unnecessary but computationally helpful queries give rise to some difficulties when one tries to directly lift the original speedup theorem to type-2.

7.1.2 Type-2 Speed-up Theorems

From the Operator Speed-up Theorem [MF72] (Theorem 101 on page 127) we can immediately conclude the following corollary, in which we replace the operator $\Theta : (\mathbf{N} \to \mathbf{N}) \to (\mathbf{N} \to \mathbf{N})$ of Theorem 101 by a functional $R : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$.

Corollary 148 *Given any computable functional $R : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, there exists a recursive function f such that,*

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

■

However, our goal is to construct a type-2 speedable functional in our programming system $\widehat{\varphi}$. Type-2 speed-up theorems vary with the nature of their speed-up factors, i.e., the speed-up factor can be either type-1 or type-2. For type-3 speed-up factors, the theorem becomes a type-2 analog of the Operator Speed-up Theorem. We will see in next section that there is no type-2 analog of the Operator Speed-up Theorem.

It is clear that our $\widehat{\varphi}$-programming system is powerful enough to code the entire class of type-1 computable functions. That is, by the definition of $\langle \widehat{\varphi}_i \rangle$, for each partial recursive function $g : \mathbf{N} \rightharpoonup \mathbf{N}$, there is a $\widehat{\varphi}$-program e such that, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{\varphi}_e(f, x) = g(x)$. Thus, any query made by such an e is unnecessary. However, according to the discussion in the previous subsection, unnecessary queries may affect the computational time. Therefore, when we try to construct a type-1 speedable function in our type-2 setting, we cannot simply cancel those $\widehat{\varphi}$-programs that make oracle queries, although the speedable type-2 functional we try to construct is meant for some type-1 computation. On the other hand, if we take all queries into account and use the same construction to construct a type-2 speedable functional, we face another difficulty in trying to make our speedable functional total. The problem is that, in general, we cannot decide whether a query is necessary or not; thus our construction will tend to be fooled by infinitely many unnecessary queries and fail to converge.

In this section we show that the above problems can be avoided by a careful translation of the original proof for the Speed-up Theorem. We will see in a moment that the definition of $\leq_{\mathbf{2}}^{*}$ (Definition 35) based on the compactness in the induced topology (Definition 30) relative to the concerned functionals plays an important role here.

Original Speed-up Theorem We sketch the original proof of the Speed-up Theorem that has been customized for our needs. A full version of the proof can be found in any of [Blu67, Blu71, Van75, Cut80, WW85, Cal88, Sei90]. We start with an intermediate theorem often called "Pseudo-Speed-up Theorem."

Theorem 149 (Pseudo-Speed-up Theorem) *Let $r : \mathbf{N} \to \mathbf{N}$ be recursive. There exists a recursive function $f_r : \mathbf{N} \to \mathbf{N}$ such that,*

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

Sketch of Proof:

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

i. $\varphi_e : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$,

ii. for every $u \in \mathbf{N}$, for all but finitely many $x \in \mathbf{N}$, $\varphi_e(0, x) = \varphi_e(u, x)$, and

iii. for every $i \in \mathbf{N}$, if $\varphi_i = \varphi_{s(e,0)}$, then $\varphi_i =^* \varphi_{s(e,i+1)}$ and $r \circ \Phi_{s(e,i+1)} <^* \Phi_i$.

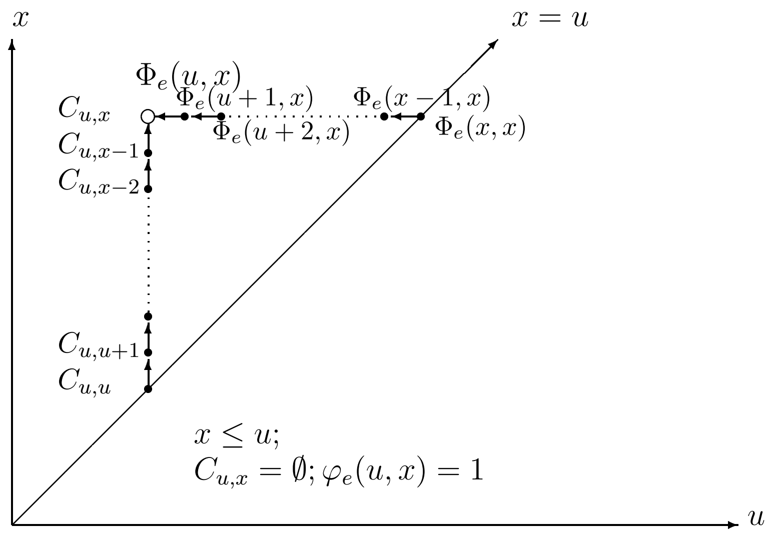

Figure 11: Construction of $C_{u,x}$

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

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

1. The previously defined sets: $C_{u,u}$, $C_{u,u+1}$, $\ldots$, $C_{u,x-1}$, and

2. the cost of computing each of $\varphi_{s(e,u+1)}(x)$, $\varphi_{s(e,u+2)}(x)$, $\ldots$, $\varphi_{s(e,x)}(x)$.

Figure 11 shows the dependence of $C_{u,x}$ on these prior sets and run times.

Precisely, for each $u, x \in \mathbf{N}$, $\varphi_e(u, x)$ and $C_{u,x}$ are defined as follows.

i. If $x \leq u$, then set $C_{u,x} = \emptyset$ and $\varphi_e(u, x) = 1$.

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

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

Claims:

1. φ_e is total.

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

3. For every $u, x_1, x_2 \in \mathbf{N}$, if $x_1 \neq x_2$, then $C_{u,x_1} \neq C_{u,x_2}$.

4. For every $u \in \mathbf{N}$, for all but finitely many $x \in \mathbf{N}$, $\varphi_e(0,x) = \varphi_e(u,x)$.

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

Proofs of the Claims:

1. For $x \leq u$, $\varphi_e(u,x)$ and $C_{u,x}$ are defined to be 1 and $\emptyset$, respective. For $x > u$, Figure 11 shows that every such point is well defined based on some finite previously defined points and cancellation sets.

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

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

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

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

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

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

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

3. Let $x_1 \neq x_2$. From the construction of the cancellation sets, it is clear that $i \in C_{u,x_1} \Rightarrow i \notin C_{u,x_2}$ and $i \in C_{u,x_2} \Rightarrow i \notin C_{u,x_1}$.

4. For every $u, x \in \mathbf{N}$, the values of $\varphi_e(0,x)$ and $\varphi_e(u,x)$ are determined by $C_{0,x}$ and $C_{u,x}$, respectively. By Claim **2**, $C_{0,x} - C_{u,x} \subseteq \{0, 1, \ldots, u-1\}$. Thus, only indices in $\{0, 1, \ldots, u-1\}$ may cause the difference between $C_{0,x}$ and $C_{u,x}$. But, by Claim **3**, each such index will be selected at most once for some x. Thus, if x is sufficiently large, all indices in $\{0, 1, \ldots, u-1\}$ will have been cancelled and will not be selected again, and hence $C_{0,x} = C_{u,x}$.

5. Suppose that $\varphi_i = \varphi_{s(e,0)}$. From Claim **4**, we already have $\varphi_i =^* \varphi_{s(e,i+1)}$. For

the other part of this claim, we assume, by contradiction, there are infinitely many x such that, $\Phi_i(x) \leq r \circ \Phi_{s(e,i+1)}(x)$. Then for some sufficiently large a with $a \geq i$, i will be selected into the cancellation set $C_{0,a}$. Hence, $\varphi_i(a) \neq \varphi_{s(e,0)}(a)$. This is a contradiction.

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

To obtain the Speed-up Theorem, we can patch the almost everywhere equality in the Pseudo-Speed-up Theorem by means of a finite table that stores the exact values of the speedable function on those exceptional points. However, the finite table cannot be uniformly constructed, and hence the proof of the Speed-up Theorem is not constructive in this sense.[1]

Type-2 Speed-up Theorems Recall the type-2 almost-everywhere relation $\leq_{\mathbf{2}}^*$ defined by Definition 35 on page 60. Consider the following two theorems:

Theorem 150 *Let $r : \mathbf{N} \to \mathbf{N}$ be any recursive function. There exists a computable functional $F_r : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that,*

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

Theorem 151 (Type-2 Speed-up Theorem) *Let $R : \mathcal{T} \times \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ be a computable functional. There exists a computable functional $F_R : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that,*

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

Theorem 150 is a lift of the statement of the original Speed-up Theorem and Theorem 151 is a life of the original Operator Speed-up Theorem. Note that since

[1] A comprehensive discussion about the constructability of the proof of the Speed-up Theorem can be found in [Van75].

Theorem 150 is a special case of Theorem 151, we consider Theorem 151 as our type-2 Speed-up Theorem. Instead of proving Theorem 151 directly, we prove a simpler result of Theorem 150. The idea used in lifting the original proof of the Speed-up Theorem for our Theorem 150 is directly applicable to prove Theorem 151.

Consider Theorem 150. We observe that $F_r = \widehat{\varphi}_i = \widehat{\varphi}_j$ in the theorem. By Definition 35, the induced topology for the type-2 relation, $r \circ \widehat{\Phi}_j \leq_{\mathbf{2}}^{*} \widehat{\Phi}_i$, is

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

Thus, if we construct F_r with $(\emptyset, x)$ as its minimal locking fragment for every $x \in \mathbf{N}$, then the induced topology for $\leq_{\mathbf{2}}^{*}$ in the theorem is the coarsest one, i.e., the topology with basic open sets: $(\!(\emptyset, 0)\!), (\!(\emptyset, 1)\!), \ldots$. The advantage is that: given any $S \subset \mathcal{T} \times \mathbf{N}$ with S noncompact in the topology $\mathbb{T}(F_r)$, we then must have that the type-0 component of the elements of S has infinitely many different values. If a $\widehat{\varphi}$-program i needs to be canceled, we thus have infinitely many chances to find it out on some type-0 inputs. We can therefore ignore the effects of the type-1 input in the computation. In other words, it is not necessary to introduce another parameter for the type-1 argument when defining the cancellation sets.

This advantage, however, disappears in its corresponding type-2 pseudo-speed-up theorem. Because, for every $\widehat{\varphi}$-program i for F_r, its *pseudo* sped-up version, $\widehat{\varphi}$-program j, does not exactly compute $\widehat{\varphi}_i$ on some small type-0 inputs, and hence $\widehat{\varphi}_i$ and $\widehat{\varphi}_j$ may define two different topologies. Thus, if we ignore the effect of the type-1 argument, the almost-everywhere relation $r \circ \widehat{\Phi}_j \leq_{\mathbf{2}}^{*} \widehat{\Phi}_i$ may fail in topology $\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_j)$. We have the following weaker theorem as our type-2 pseudo-speed-up theorem, in which the compactness is not considered. Nevertheless, this weaker type-2 pseudo-speed-up theorem will be sufficient for our proof of Theorem 150.

Theorem 152 (Type-2 Pseudo-Speed-up Theorem) *Let $r : \mathbf{N} \to \mathbf{N}$ be any recursive function. There exists a computable functional $F_r : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ such that, for every $\widehat{\varphi}$-program i for F_r, there is another $\widehat{\varphi}$-program j such that,*

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

Sketch of Proof:

Fix a recursive function $r : \mathbf{N} \to \mathbf{N}$. Recall the s-m-n and recursion theorems on the type-0 argument introduced in Section 1.5. Let s be an s-1-2 function such that, for every $e, u, x \in \mathbf{N}$ and $f \in \mathcal{T}$, $\widehat{\varphi}_{s(e,u)}(f,x) = \widehat{\varphi}_e(u,f,x)$. We shall construct, by the recursion theorem, a $\widehat{\varphi}$-program e that is similar to the φ-program in Theorem 149, that is,

i. $\widehat{\varphi}_e : \mathbf{N} \times \mathcal{T} \times \mathbf{N} \to \mathbf{N}$.

ii. For every $u \in \mathbf{N}$, there exists $n_0 \in \mathbf{N}$ such that, for every $x > n_0$ and $f \in \mathcal{T}$, we have $\widehat{\varphi}_e(0,f,x) = \widehat{\varphi}_e(u,f,x)$.

iii. For every $i \in \mathbf{N}$, if $\widehat{\varphi}_i$ computes $\widehat{\varphi}_{s(e,0)}$, then there exists $n_0 \in \mathbf{N}$ such that, if $x \in \mathbf{N}$ with $x > n_0$, then for every $f \in \mathcal{T}$, $\widehat{\varphi}_i(f,x) = \widehat{\varphi}_{s(e,i+1)}(f,x)$ and $\widehat{\Phi}_i(f,x) > r \circ \widehat{\Phi}_{s(e,i+1)}(f,x)$.

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

i. If $x \leq u$, then let $C_{u,x} = \emptyset$ and $\widehat{\varphi}_e(u,f,x) = 1$.

ii. If $x > u$, then define $C_{u,x}$ by:

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

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

$$\widehat{\varphi}_e(u, f, x) = 1 + \mathsf{max}(\{\widehat{\varphi}_i(f_0, x) \mid i \in C_{u,x}\}). \tag{46}$$

Consider the five claims in the proof of Theorem 149. We have six claims for the present theorem as follows.

1. $\widehat{\varphi}_e$ is total on $\mathbf{N} \times \mathcal{T} \times \mathbf{N}$.

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

3. For every $u, x_1, x_2 \in \mathbf{N}$, if $x_1 \neq x_2$, then $C_{u,x_1} \neq C_{u,x_2}$.

4. For every $u \in \mathbf{N}$, for all but finitely many $x \in \mathbf{N}$, and for every $f \in \mathcal{T}$, $\widehat{\varphi}_e(0, f, x) = \widehat{\varphi}_e(u, f, x)$.

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

6. If i is a $\widehat{\varphi}$-program for $\widehat{\varphi}_{s(e,0)}$, then, for all but finitely many $x \in \mathbf{N}$ and for all $f \in \mathcal{T}$, the OTM $\widehat{M_i}$, on (f, x), does not make any oracle query.

Claims 2, 3, 4, and 5 can be proved by exactly the same arguments for Theorem 149. For claim 1, it is clear that the extra clause

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

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

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

Proof of Theorem 150: According to the construction of $\widehat{\varphi}_e$ in Theorem 152, for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{\varphi}_e(0, f, x) = \widehat{\varphi}_e(0, f_0, x)$. It follows that $(\emptyset, x)$ is the minimal locking fragment of $\widehat{\varphi}_{s(e,0)}$ on every $(f, x) \in \mathcal{T} \times \mathbf{N}$. Let $\widehat{\varphi}_i = \widehat{\varphi}_{s(e,0)}$ and $j = s(e, i+1)$. Note that $\widehat{\varphi}_i =^*_{\mathbf{2}} \widehat{\varphi}_j$ and $r \circ \widehat{\Phi}_j \leq^*_{\mathbf{2}} \widehat{\Phi}_i$ does not hold in general in the Type-2 Pseudo-Speed-up Theorem, because $(\!(\emptyset, x)\!)$ is not necessarily a basic open set when x is small. Nevertheless, this is not important. Consider the following exception set

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

E may not be compact in topology $\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_j)$, but, according to the construction, $\{x | (f, x) \in E\}$ is finite. Thus, we can have a patched $\widehat{\varphi}$-program, j', that will check the type-0 input first. If the type-0 argument has a value in $\{x | (f, x) \in E\}$, then $\widehat{\varphi}$-program j' outputs the correct value stored in some finite table. In such a way, the type-1 input will not affect the result, and hence the minimal locking fragment becomes $(\emptyset, x)$. On the other hand, if type-0 argument $x \notin \{x | (f, x) \in E\}$, then $\widehat{\varphi}$-program j' starts running $\widehat{\varphi}$-program j.

Similarly, the exception set

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

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

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

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

Finally, we consider the problem pointed out in the beginning of this subsection. That is, there may exist some best $\widehat{\varphi}$-program for $\widehat{\varphi}_{s(e,0)}$ using some unnecessary queries to escape from being canceled which is supposed to be made by (46). This is possible because we replace the actual type-1 input by f_0 for every $\widehat{\varphi}$-program being tested, and hence we do not know the program's behavior on actual $f \in \mathcal{T}$. Clearly, by Claim 6 in the proof of Theorem 152, we do not have to worry about this problem, because any program that will make any query on some inputs does not compute our speedable functional.

This completes the proof of Theorem 150. $\square$

7.2 The Type-2 Operator Anti-speed-up Theorem

In the previous section we established two speed-up theorems. In Theorem 150 we consider type-1 speed-up factors and the proof is indeed lifted from a proof for the original Speed-up Theorem. In Theorem 151 we consider type-2 speed-up factors and it is lifted from the original Operator Speed-up Theorem. In this section we consider a type-2 analog of the original Operator Speed-up Theorem. That is, we will try to explore a speed-up phenomenon when the speed-up factor is type-3. Clearly, a proof to such a type-2 operator speed-up theorem needs a general type-2 s-m-n and a type-2 recursion theorem, which we don't have (see Section 1.5). In fact, the type-2 analog of the Operator Speed-up Theorem does not exist. In the following, we present a theorem that is stronger than the direct negation of the type-2 analog of the Operator Speed-up Theorem.

By "an effective type-2 operator" we mean a computable type-3 functional [Kle60, Kle62] of type $(\mathcal{T} \times \mathbf{N} \to \mathbf{N}) \to (\mathcal{T} \times \mathbf{N} \to \mathbf{N})$ with inputs restricted to computable total type-2 functionals. Thus, we can think up that an effective type-2 operator is given by a $\widehat{\varphi}$-program that takes a total $\widehat{\varphi}$-program as its input and outputs another total $\widehat{\varphi}$-program. Our next theorem asserts that there is an effective type-2 operator $\widehat{\Theta}$ such that, for *every* total $\widehat{\varphi}$-program e, there is no $\widehat{\Theta}$-sped up version for e. Recall from Definition 37 the type-2 analogs of "for all but finitely many" and "infinitely often" quantifiers, $\overset{\infty}{\forall}_{\mathbf{2}}$ and $\overset{\infty}{\exists}_{\mathbf{2}}$.

Theorem 153 (Type-2 Operator Anti-Speed-up Theorem) *There is a type-2 effective operator* $\widehat{\Theta} : (\mathcal{T} \times \mathbf{N} \to \mathbf{N}) \to (\mathcal{T} \times \mathbf{N} \to \mathbf{N})$ *such that, for every computable functional,* $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$*, we have*

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

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

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

Clearly, such $\widehat{\Theta}$ is a type-2 effective operator.

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

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

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

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

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

Note that, a direct negation of the type-2 analog of the Operator Speed-up Theorem is that:

Corollary 154 *There is a type-2 effective operator $\widehat{\Theta} : (\mathcal{T} \times \mathbf{N} \to \mathbf{N}) \to (\mathcal{T} \times \mathbf{N} \to \mathbf{N})$ such that, for all computable $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, we have*

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

Proof: Clearly, this follows Theorem 153 immediately. □

7.3 Query-optimum Programs

Since the number of queries made throughout the course of the computation is a dynamic computational resource that can be seen only in type-2 computations, we now move to consider the query-complexity as a complexity measure. We are interested in whether, for any type-2 computable functional, there is a *query-optimal program* for it, or whether there is a speed-up phenomenon for query-complexity. It turns out that the query-complexity fails to satisfy Blum's two axioms. Consequently, the technique for the original proof of the Speed-up Theorem cannot be used for proving a query-complexity speed-up theorem. Thus, we need a new approach to for this problem.

As we saw in the previous section, there are functionals whose computations always require some *unnecessary queries* (Definition 146). So it does not seem sensible to demand that a *query-optimal* program makes only necessary queries. We thus consider some alternative notions of "*query-optimum programs*" in this section. In the context of the speed-up phenomenon, we find that the stronger the notion of *query-optimum*, the easier it is to achieve a speed-up theorem. That is, it is more difficult to obtain a query-optimum $\widehat{\varphi}$-program when the notion of query-optimum becomes stronger. We do not know how far can one weaken the notion of query-optimum programs so that the speed-up phenomenon disappears and yet the notion of query-optimum programs is still nontrivial. We give a partial results in this direction.

Definition 155 *Let e be a $\widehat{\varphi}$-program. Define $Q : (\mathbf{N} \times \mathbf{N} \times \mathcal{T} \times \mathbf{N}) \to \mathcal{F}$ by: $Q(s, e, f, x) = \tau$, where $\tau \in \mathcal{F}$ with $\tau \subset f$ and $\mathsf{dom}(\tau)$ is the collection of queries made during the course of the computation of OTM $\widehat{M_e}$ on (f, x) in s steps.* ■

Note that $Q(s, e, f, x)$ is monotone in s, i.e., for every $s \in \mathbf{N}$

$$Q(s, e, f, x) \subseteq Q(s + 1, e, f, x).$$

Definition 156 *Let $e, x \in \mathbf{N}$ and $f \in \mathcal{T}$. We say that $Q(\cdot, e, f, x)$ is defined in the limit if and only if there exist $s \in \mathbf{N}$ and $\tau \in \mathcal{F}$ such that, for every $s' \geq s$,*

$$Q(s, e, f, x) = Q(s', e, f, x) = \tau.$$

In the case above, we write $\lim\limits_{s\to\infty} Q(s, e, f, x) \downarrow = \tau$. ■

We wish to treat the set of queries an OTM made as a dynamic computational recourse, but this is very far from satisfying Blum's two complexity measure axioms. That is, $\lim\limits_{s\to\infty} Q(s, e, f, x) \downarrow$ does not mean that $\widehat{\varphi}_e(f, x)$ converges, and hence Blum's first axiom fails. Moreover, for some $\sigma \in \mathcal{F}$ with $\sigma \subset f$, $\lim_{s\to\infty} Q(s,e,f,x) = \sigma$ is not necessarily decidable, and hence Blum's second axiom fails too.

Let $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be computable. Since we mean to characterize some $\widehat{\varphi}$-programs $e_1, e_2, \ldots, e_n$ that compute the same functional F in terms of query-complexity, the relative topology $\mathbb{T}(\cdots)$ defined in Chapter 2 cannot serve as the induced topology for the compactness. This is because $\widehat{\varphi}_{e_1} = \widehat{\varphi}_{e_2} = \cdots \widehat{\varphi}_{e_n} = F$ and thus $\mathbb{T}(\widehat{\varphi}_{e_1}, \widehat{\varphi}_{e_2}, \ldots, \widehat{\varphi}_{e_n}) = \mathbb{T}(F)$, and hence the complexity of unnecessary queries after locking fragments cannot be reflected in the topology $\mathbb{T}(F)$. On the other hand, the finest topology, $\mathbb{T} \times \mathbb{N}$, works for our purpose but it is not suitable for the same reasons discussed in Chapter 2. We thus introduce another family of topologies as follows.

Definition 157 *Let $e_1, e_2, \ldots, e_n \in \mathbf{N}$. Let $B(e_1, e_2, \ldots, e_n) \subseteq \mathcal{T} \times \mathbf{N}$ such that, $(\sigma, a) \in B(e_1, e_2, \ldots, e_n)$ if and only if, for some $f \in \mathcal{T}$, we have that, for each $i = 1, \ldots, n$,*

$$\lim_{s\to\infty} Q(s, e_i, f, a)\downarrow = \sigma_i,$$

and $\sigma = \bigcap_{1\le i\le n} \sigma_i$. Let $\mathbb{Q}(e_1, \ldots, e_n)$ be the topology on $\mathcal{T} \times \mathbf{N}$ with basic open sets as

$$\mathcal{B} = \left\{ (\!(\sigma, x)\!) \;\middle|\; (\sigma, x) \in B \right\}.$$

■

Since a $\widehat{\varphi}$-program e may make unnecessary queries outside the domains of its minimal locking fragments, it follows that the topology $\mathbb{Q}(e)$ may be finer than the topology $\mathbb{T}(\widehat{\varphi}_e)$ which is defined by the minimal locking fragments of $\widehat{\varphi}_e$. That is, $\mathbb{T}(\widehat{\varphi}_e) \subseteq \mathbb{Q}(e)$. In other words, if e is a $\widehat{\varphi}$-program for $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, then

$\mathbb{T}(F) \subseteq \mathbb{Q}(e)$. On the other hand, if $\mathbb{Q}(e) \subset \mathbb{T}(F)$, then we can conclude that e cannot be a $\widehat{\varphi}$-program for F. In general, we have

$$\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_j) \subseteq \mathbb{Q}(i, j).$$

Definition 158 *Let $\widehat{\varphi}_i = \widehat{\varphi}_j$. Define $Q_{[i<j]}, Q_{[i=j]}$, and $Q_{[i>j]}$ by:*

$$\begin{aligned}
Q_{[i<j]} &= \left\{ (f,x) \;\middle|\; \mathsf{card}(\lim_{s\to\infty} Q(s,i,f,x)) < \mathsf{card}(\lim_{s\to\infty} Q(s,j,f,x)) \right\}. \\
Q_{[i=j]} &= \left\{ (f,x) \;\middle|\; \mathsf{card}(\lim_{s\to\infty} Q(s,i,f,x)) = \mathsf{card}(\lim_{s\to\infty} Q(s,j,f,x)) \right\}. \\
Q_{[i>j]} &= \left\{ (f,x) \;\middle|\; \mathsf{card}(\lim_{s\to\infty} Q(s,i,f,x)) > \mathsf{card}(\lim_{s\to\infty} Q(s,j,f,x)) \right\}.
\end{aligned}$$

■

Definition 159 (Query Speed-up) *Let $\widehat{\varphi}_i = \widehat{\varphi}_j$.*

1. *We say that j is a **weakly** query sped-up version of i if and only if*

   $$Q_{[i<j]} \text{ is compact and } Q_{[i>j]} \text{ is noncompact in } \mathbb{Q}(i,j).$$

   *If j is a **weakly** query sped-up version of i, we write $\widehat{\varphi}_i \preceq^*_Q \widehat{\varphi}_j$.*

2. *We say that j is a **strongly** query sped-up version of i if and only if*

   $$Q_{[i<j]} \cup Q_{[i=j]} \text{ is compact in } \mathbb{Q}(i,j).$$

   *If j is a **strongly** query sped-up version of i, we write $\widehat{\varphi}_i \preceq\!\!\!\prec^*_Q \widehat{\varphi}_j$.* ■

Suppose that we are to write a new $\widehat{\varphi}$-program j for $\widehat{\varphi}_i$. Intuitively, if the new $\widehat{\varphi}$-program j can improve the query-complexity on a "large" (noncompact) set of inputs and leaves a "small" (compact) set of inputs on which j queries more than i does, we say that j is a *weakly* query sped-up version of i. Note that, we do not care how many inputs on which the query-complexity remains unchanged. Since $\mathbb{T}(\widehat{\varphi}_j) \subseteq \mathbb{Q}(i,j)$, it follows that, if $Q_{[i<j]}$ is compact in $\mathbb{Q}(i,j)$, so is it in $\mathbb{T}(\widehat{\varphi}_j)$, and hence we can patch $\widehat{\varphi}$-program j on $Q_{[i<j]}$ so that the patched $\widehat{\varphi}$-program will never query more that i does. Whereas, if $\widehat{\varphi}$-program j is a *strongly* query sped-up version of i, then the set on which j does not improve its query-complexity must be "small" (compact). Note that, if $Q_{[i<j]} \cup Q_{[i=j]}$ is compact in $\mathbb{Q}(i,j)$, then $Q_{[i>j]}$ must be noncompact in $\mathbb{Q}(i,j)$. Clearly, for $\widehat{\varphi}_i = \widehat{\varphi}_j$, we have

$$\widehat{\varphi}_i \preceq\!\!\preceq^*_Q \widehat{\varphi}_j \implies \widehat{\varphi}_i \preceq^*_Q \widehat{\varphi}_j.$$

Definition 160 (Query-optimum $\widehat{\varphi}$-programs) *Let $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be a computable functional. Let e be a $\widehat{\varphi}$-program for F.*

1. *We say that e is an* ***absolute*** *query-optimum $\widehat{\varphi}$-program for F if and only if, on every $(f,x) \in \mathcal{T} \times \mathbf{N}$, the OTM $\widehat{M}_e$ does not make unnecessary queries during the course of computing $\widehat{\varphi}_e(f,x)$.*

2. *We say that e is a* ***strong*** *query-optimum $\widehat{\varphi}$-program for F if and only if there is no $\widehat{\varphi}$-program i for F such that, $\widehat{\varphi}_e \preceq^*_Q \widehat{\varphi}_i$.*

3. *We say that e is a* ***weak*** *query-optimum $\widehat{\varphi}$-program for F if and only if there is no $\widehat{\varphi}$-program i for F such that, $\widehat{\varphi}_e \preceq\!\!\preceq^*_Q \widehat{\varphi}_i$.* ■

Suppose that e is a $\widehat{\varphi}$-program for $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$. Thus, if e is an absolute query-optimum $\widehat{\varphi}$-program for F, then it is also a strong query-optimum $\widehat{\varphi}$-program for F. Likewise, if e is an strong query-optimum $\widehat{\varphi}$-program for F, then it is also a weak query-optimum $\widehat{\varphi}$-program for F. Moreover, if $\mathbb{T}(\widehat{\varphi}_e) = \mathbb{Q}(e)$, then e is an absolute query-optimum $\widehat{\varphi}$-program for $\widehat{\varphi}_e$.

Fix a complexity measure satisfying Blum's axioms. Then, the type-1 speed-up theorem says that, roughly, there is a total computable function without optimal program for it. Although query-complexity does not satisfy the Blum's axioms, each of the three versions of query-optimum programs gives rise to the following questions:

1. Does every computable functional $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ always have a query-optimum $\widehat{\varphi}$-program for it?

2. If the answer to the first question is negative, then can we construct the query-speed-able functional?

3. If a given $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ does have a query-optimum $\widehat{\varphi}$-programs for it, then can we uniformly construct a query-optimum program for F from an arbitrary $\widehat{\varphi}$-program for F?

4. Suppose there is no query-optimum $\widehat{\varphi}$-program for $\widehat{\varphi}_{e_0} : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$. Then, can we effectively construct an infinitely sequence of query-sped-up version of $\widehat{\varphi}$-programs from e_0? That is, can we have $e_0, e_1, e_2, \ldots$ such that, for each $i \in \mathbf{N}$, e_{i+1} is a query-sped-up version of e_i?

We positively answer the second question for *absolute* and *strong* versions of query-optimum programs. Thus, a negative answer to the first question follows immediately. In fact, the speedable functional we constructed for the second question is very simple; this is in contrast to the nonconstructivity of the original speed-up theorem. However, we do not know if there is a computable $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ that can forever strongly speed-up, i.e., we do not know if there is a computable $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ without a weak query-optimum $\widehat{\varphi}$-program. To the third question, Theorem 165 gives a negative answer. However, the answer to the fourth question is still open. That is, given that $\widehat{\varphi}_{e_0} : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ is speedable, we do not know if there is an effective way to construct a sequence from e_0 such that,

$$\widehat{\varphi}_{e_0} \preceq^*_Q \widehat{\varphi}_{e_1} \preceq^*_Q \widehat{\varphi}_{e_2} \preceq^*_Q \cdots, \quad \text{or} \quad \widehat{\varphi}_{e_0} \underline{\prec\!\!\prec}^*_Q \widehat{\varphi}_{e_1} \underline{\prec\!\!\prec}^*_Q \widehat{\varphi}_{e_2} \underline{\prec\!\!\prec}^*_Q \cdots.$$

To begin, we first introduce a functional $K : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ that is used repeatedly in the following. Let $\varphi_x(x) \downarrow_s$ be true if and only if the Turing Machine M_x, on x, halts in s steps.

Definition 161 *Define $K : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ by*

$$K(f, x) = \begin{cases} \varphi_x(x) + 1 & \text{if } \varphi_x(x) \downarrow_s, \text{ where } s = \sum_{i=0}^{f(0)} f(i); \\ 0 & \text{otherwise,} \end{cases}$$

■

Let $\mathbf{K}$ denote the set $\{x|\varphi_x(x) \downarrow\}$. For any $(f, x) \in \mathcal{T} \times \mathbf{N}$, if $x \notin \mathbf{K}$, then any $\widehat{\varphi}$-program that computes K, on input (f, x), may ask arbitrarily many unnecessary queries, depending on the value of $f(0)$. In fact, it is easy to see that there is no $\widehat{\varphi}$-program for K with all unnecessary queries removed. Even if we lower the standard of being a query-optimum program, we still cannot have either a strong query-optimum $\widehat{\varphi}$-program or a weak query-optimum $\widehat{\varphi}$-program for K. We state and prove these in the following theorems.

Theorem 162 *K has no absolute query-optimum $\widehat{\varphi}$-program.*

Proof: We show that from an absolute query-optimum $\widehat{\varphi}$-program for K one can construct a solution to the halting problem.

Suppose, by contradiction, $\widehat{\varphi}_e$ is an absolute query-optimum $\widehat{\varphi}$-program for K. Clearly, if $x \notin \mathbf{K}$, then for any $f \in \mathcal{T}$, $K(f, x) = 0$. On the other hand, if $x \in \mathbf{K}$, there must be two distinct $f, g \in \mathcal{T}$ such that $\widehat{\varphi}_e(f, x) \neq \widehat{\varphi}_e(g, x)$. Thus, some queries

```
Program
    input f : T, x : N;
    if x = i and f(0) is even
        then output 0;
        else output φ̂_e(f, x);        /* by running φ̂-program e on (f, x) */
End program
```

Figure 12: A Strongly Sped-up Version of $\widehat{\varphi}_e$

to the oracles f and g must be made. Therefore, given any $x \in \mathbf{N}$, we can run $\widehat{\varphi}_e(\emptyset^{\sim 0}, x)$ first, then if no queries had ever made during the course of computation, we know that $x \notin \mathbf{K}$. Hence, $\mathbf{N} - \mathbf{K}$ is recursively enumerable, a contradiction. $\square$

Theorem 163 *K has no strong query-optimum $\widehat{\varphi}$-program.*

Proof: Suppose, by contradiction, $\widehat{\varphi}_e$ is a strong query-optimum $\widehat{\varphi}$-program for K. Let $i \notin \mathbf{K}$ such that, for any $f \in \mathcal{T}$, the OTM, $\widehat{M_e}$, on (f, i), will make oracle queries at points $0, 1, 2, \ldots, f(0)$. Such i must exist, otherwise the halting problem can be solved by a similar argument given for the proof of Theorem 162. We construct a *weakly* sped-up version of e as shown in Figure 12.

Let the index of the $\widehat{\varphi}$-program in Figure 12 be p. Clearly, if $x \neq i$ or if $f(0)$ is odd, then the two $\widehat{\varphi}$-programs, e and p, will perform the same computation, and hence the two make the same oracle queries. The two computations differ on the following set:

$$\left\{ (f, i) \;\middle|\; f(0) \text{ is even} \right\}. \tag{47}$$

Clearly, the set above is noncompact in $\mathbb{Q}(e, p)$. According to the program p and the assumption on i, the following set is the same as (47),

$$\left\{ (f, i) \;\middle|\; \mathsf{card}(\lim_{s \to \infty} Q(s, e, f, i)) > \mathsf{card}(\lim_{s \to \infty} Q(s, p, f, i)) \right\}.$$

Thus, $Q_{[e>p]}$ is noncompact in $\mathbb{Q}(e,p)$. Moreover, $Q_{[e<p]} = \emptyset$, and hence $Q_{[e<p]}$ is compact in $\mathbb{Q}(e,p)$. It follows that p is a weakly sped-up version of e, i.e., $\widehat{\varphi}_e \preceq^*_Q \widehat{\varphi}_p$. Therefore, e cannot be a strong query-optimum $\widehat{\varphi}$-program for K. $\square$

Theorem 164 *There is a weak query-optimum $\widehat{\varphi}$-program for K.*

Sketch of Proof: It is impossible to strongly speed-up any $\widehat{\varphi}$ forever. Otherwise, we can reach two $\widehat{\varphi}$ programs e and p such that, $\widehat{\varphi}_e = \widehat{\varphi}_p = K$ and $\widehat{\varphi}_e \underline{\prec\prec}^*_Q$. Then, we can compare the queries made during the course of the computations between $\widehat{\varphi}_e$ and $\widehat{\varphi}_p$ to solve the halting problem. Therefore, a weak query-optimum $\widehat{\varphi}$-program for K must exist. We skip details of the proof. $\square$

However, we do not know if there is a computable type-2 functional such that, there is no weak query-optimum $\widehat{\varphi}$-program for it. In other words, we do not if there is a speed-up phenomenon associated with the notion of weak query-optimum.

Suppose we know that a query-optimum program for $\widehat{\varphi}_e$ does exist. In general, there is no effective way to construct a query-optimum version of e in any strength, i.e., absolute, strong, and weak query-optimum. We state this in the following theorem, which is the strongest version in the sense that the weakest version of query-optimum programs is considered.

Theorem 165 *There is no recursive $r : \mathbf{N} \to \mathbf{N}$ such that, for every $\widehat{\varphi}$-program e, if $\widehat{\varphi}_e$ is total and there is a weak query-optimum program for it, $r(e)$ is a weak query-optimum $\widehat{\varphi}$-program for $\widehat{\varphi}_e$.*

Proof: By contradiction, suppose such recursive $r : \mathbf{N} \to \mathbf{N}$ exists. Consider the following program shown in Figure 13. Suppose that, by the recursion theorem, the index of the $\widehat{\varphi}$-program is e.

```
Program e
    input f : T, x : N;
    Q ⟵ Q(f(0), r(e), f, x);
    if φ̂_{r(e)}(f, x) converges in f(0) steps
      then output f(max(dom(Q)) + 1);
      else output f(1);
End program e
```

Figure 13: A $\widehat{\varphi}$-program e for Theorem 165

According to the construction, it is clear that e is a total $\widehat{\varphi}$-program. By assumption, $r(e)$ is a weak query-optimum version of e. On any $(f, x) \in \mathcal{T} \times \mathbf{N}$, the OTM $\widehat{M}_{r(e)}$ will never halt in $f(0)$ steps, otherwise the OTM $\widehat{M}_{r(e)}$ does not compute $\widehat{\varphi}_e$. This is because if the OTM $\widehat{M}_{r(e)}$ on (f, x) halts in $f(0)$ steps, Q must contain the complete collection of queries made by the OTM $\widehat{M}_{r(e)}$ on (f, x). But $\widehat{\varphi}_e(f, x)$ is the value of f at a point not in Q.

Thus, we conclude that for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{\varphi}_e(f, x) = f(1)$. But if that is the case, $f(0)$ will not be queried by the OTM $\widehat{M}_{r(e)}$, namely, the value of $f(0)$ does not affect the computation of $\widehat{M}_{r(e)}$ at all. Thus, if $f(0)$ is sufficiently large, $\widehat{\varphi}_{r(e)}(f, x)$ will converge in $f(0)$ steps, and hence $\widehat{\varphi}_e(f, x) = f(\mathsf{max}(\mathsf{dom}(Q)) + 1)$. Thus, the OTM $\widehat{M}_{r(e)}$ does not compute $\widehat{\varphi}_e$.

Therefore, $r(e)$ cannot be a weak query-optimum version of $\widehat{\varphi}$-program e. □

Chapter 8

Conclusion

This thesis presents a machine based notion of type-2 complexity classes in search of a general type-2 complexity theory. From such notion, we obtain some complexity results in type-2 along the lines of classical complexity theory. We conclude these results with a clearer picture of what type-2 complexity theory should be. This provides a partial answer to Constable's plausible question asked almost three decades ago. We also point out some possible future studies and applications of our results in this final chapter.

8.1 What We Have Done

We use the Oracle Turing Machine and its time complexity as our type-2 computing apparatus and complexity measure in our study. We define a class of functions, $\mathbf{T_2TB}$, such that, each $\beta \in \mathbf{T_2TB}$ determines a *limit functional* with some special properties. We show that these extra properties are needed for our clocking purpose. Then, we define a type-2 complexity class $\mathbf{C}(\beta)$, for every $\beta \in \mathbf{T_2TB}$. The notation $\mathbf{C}(\beta)$ reflects the same ideas involved in defining classical type-1 *DTIME*(t). A key idea in defining $\mathbf{C}(\beta)$ is the "almost-everywhere relation" which allows an $F \in \mathbf{C}(\beta)$ to have a "small amount" of inputs on which the computational complexity of F is not bounded by β. We carefully study the topological nature of type-2 computation and understand that the notion of "small amount" in type-2 should be the notion of "compact sets" in an appropriate topology and such a compact set must conduce to only finitely many computations of the concerned functional. We define such an appropriate topology by $\mathbb{T}(F)$ that is the induced topology determined by the participated functional, F. We also extend the definition to $\mathbb{T}(A)$ where A is a finite set of type-2 computable functionals.

We follow Seth's suggestion in [Set94] with the goal of providing type-2 analogs of several classic type-1 complexity theorems such as Union, Gap, Compression, and Speed-up Theorems. However, after we give a precise definition of type-2 complexity classes and closely examine the idea behind those theorems, we find that not every type-1 complexity theorem can be easily obtained by lifting its original proof. We discover that the computational behaviors in type-1 and type-2 are very different. Thus, we prove our own versions of type-2 Union, Gap, Compression, and Speed-up Theorems. We also point out that a type-2 analog of the Honesty Theorem is of no importance in type-2 computation because of the type-2 Compression Theorem.

One plausible application of our type-2 Union Theorem under the answer-length-cost OTM model is the type-2 big-O notation. For every $\beta \in \mathbf{T}_2\mathbf{TB}$, $\mathbf{O}(\beta)$ turns out to be a type-2 complexity class, i.e., there is an $\alpha \in \mathbf{T}_2\mathbf{TB}$ such that, $\mathbf{O}(\beta) = \mathbf{C}(\alpha)$.

The number of oracle queries made during the course of the computation is a special complexity measure in type-2 computation. In many applications such as machine learning, the query-complexity has been seriously considered. However, query-complexity fails to meet Blum's two axioms, which adds to the difficulty of the study. Nevertheless, we gave three notions of query-optimum programs to see how well we can have an optimum program for a computable type-2 functional in general. A few phenomena in query-complexity have studied, which can be seen as query analogs of the speed-up phenomenon.

8.2 Possible Applications

We believe that our research has given a clearer picture of the complexity concern for many problems that bear the type-2 nature such as machine learning, real number computation, functional programming, and so on.

For example, our clocking scheme can immediately apply to the framework of machine learning. Feasible learnability has been a long concern in the area. Although we point out that $\mathbf{BFF}_2$ fail to be a type-2 complexity class, it is easy to see that $\mathbf{BFF}_2$ can be described as a union class determined by a uniform $\langle\beta_i\rangle$. If a class of concepts is **BFF**-learnable, then we can find a learner clocked by some β_i to learn the concepts. For example, we can restate one of Angluin's results of [Ang87] in machine learning as follows: *Deterministic Finite Automaton* (DFA) is POTM learnable but not **BFF** learnable.[1] This is because the number of queries made to learn DFA is unbounded, but each β_i in $\langle\beta_i\rangle$ that determines $\mathbf{BFF}_2$ must be bounded in terms of the length of minimal locking fragments. Moreover, we can extend the idea to any $\beta \in \mathbf{T}_2\mathbf{TB}$. We say that a class of concepts is β-learnable if there is a machine clocked by β that can learn the concepts.

[1]This result is based on a learning model called *adequate-teacher* learning model in which the adequate teacher is capable to answer *equivalence* and *membership* queries [Ang87].

8.3 Open Problems and Possible Future Studies

Here we point out some open problems that we fail to solve in this thesis. Also, we suggest a possible direction for extending our work to all fine types in future studies.

1. In type-1, time-constructibility turns out to be a useful property for time bounds. Many interesting theorems depend on this properties such as *hierarchy theorems*. We do not know what is a proper analog of time-constructibility for our type-2 time bounds.

2. The conditions for the Type-2 Union Theorem under the answer-length-cost model are rather involved. The theorem is not at all robust to those conditions as we have seen that a slight change of the conditions can easily break the theorem. Strong $\mathcal{F}$-monotonicity is a welcome property for type-2 time bounds to have, with which our clocking scheme will not shrink the budget during the computation. Thus, it will be nice if we can construct a *strong* type-2 time bound in the Type-2 Union Theorem. However, we are not able to do so; even when $\langle \beta_i \rangle$ is given strong. We strongly feel that it is impossible to construct a strong type-2 time bound in the union theorem. It is because that, to maintain a lower bound for the α in the theorem, we need directly invalidate the priority arguments. However, we are not able to prove this conjecture at the moment.

   Conjecture: There is a uniform, ascending, and strong $\langle \beta_i \rangle$ such that, if there is $\alpha \in \mathbf{T}_2\mathbf{TB}$ such that $\mathbf{C}_{\mathbf{A}}(\alpha) = \mathbf{C}_{\mathbf{A}}(\langle \beta_i \rangle)$, then α is not strong. ■

3. Under the unit-cost model, it is much easier to construct a union class that is not a type-2 complexity class. Thus, we can strengthen the condition on $\langle \beta_i \rangle$ but the union theorem still does not hold. The hypothesis in Corollary 132

seems to be the strongest non-trivial one. We do not know if we can move one step further as in the following conjecture.

Conjecture: There exist a total computable functional $H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ and a uniform, ascending, useful, and strongly convergent $\langle \beta_i \rangle$ such that, for every $i \in \mathbf{N}$, $F_{\beta_i} \le H$ and $\mathbf{C}_{\mathbf{U}}(\langle \beta_i \rangle)$ is not a type-2 complexity class. ■

4. In the context of query complexity, we define three notions for query-optimum programs. We observe that it is more difficult to obtain a query-optimum program when the notion of query-optimum becomes stronger. For the two stronger versions of the definitions of query-optimum programs, we have a computable type-2 functional that has no query-optimum programs for it. However, we do not know if there is a computable type-2 functional that has no *weak query-optimum* program for it. In other words, we do not know if the speed-up theorem holds under our weakest notion of query-optimum programs.

Future Studies: An obvious direction to extend our work is to define type-3 and above complexity classes in terms of (*i*) higher type time bounds, (*ii*) higher type almost-everywhere relations, and (*iii*) higher type clocking schemes. However, as we have seen that the ideas in type-2 are not obtained by straightforward lifting their counterparts in type-1, we do not think it will be a trivial work to extend our complexity results to all finite types. If we can provide reasonable notions for (*i*), (*ii*), and (*iii*), then we should be able to extend our work to type-3 and above. However, we have to face another problem. There is little disagreement between the notions of type-2 computability. But this is not the case in type-3 and above. Thus, we have to choose a formalism for the notion of computability at all simply finite types. One possible choice is the notion of higher type computability in the theory of *countable functionals* led by Kleene, Kreisel, et al. Another direction is to choose a programming formalism such as PCF, and then explore the complexity properties of PCF-computable functionals.

Appendix A

Notations and Conventions

1. $\mathbf{N} \triangleq$ The set of all natural numbers.

2. $\mathcal{R} \triangleq \mathbf{N} \to \mathbf{N}$, the set of all recursive functions.

3. $\mathcal{P} \triangleq \mathbf{N} \rightharpoonup \mathbf{N}$, the set of all partial functions.

4. $\mathcal{T} \triangleq \mathbf{N} \to \mathbf{N}$, the set of all total functions.

5. $\mathcal{PR} \triangleq \mathbf{N} \rightharpoonup \mathbf{N}$, the set of all partial recursive functions.

6. $\mathcal{F} \triangleq$ The set of all finite functions from $\mathbf{N}$ to $\mathbf{N}$. That is, for each $\sigma \in \mathcal{F}$, the domain of σ is a finite subset of $\mathbf{N}$. Unless stated otherwise, τ and σ range over $\mathcal{F}$.

7. $\mathsf{dom}(\sigma) \triangleq$ The domain of σ.

8. $\mathsf{card}(\sigma) \triangleq$ The cardinality of $\mathsf{dom}(\sigma)$.

9. $\langle \cdot, \cdot \rangle : \mathbf{N} \times \mathbf{N} \to \mathbf{N} \triangleq$ A fixed pairing function. That is, $\langle \cdot, \cdot \rangle$ is a fixed computable bijection between $\mathbf{N} \times \mathbf{N}$ and $\mathbf{N}$.

10. $\mathbb{N} \triangleq$ The discrete topology on the space $\mathbf{N}$.

11. $\mathbb{T} \triangleq$ The Baire topology on the space $\mathcal{T}$.

12. $\mathbb{T} \times \mathbb{N} \triangleq$ The product topology of $\mathbb{T}$ and $\mathbb{N}$.

13. $\mathbb{T}(F) \triangleq$ The induced topology of $\mathbb{T}\times\mathbb{N}$ determined by the continuous functional $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$.

14. $\leq_{\mathbf{2}}^{*} \triangleq$ The type-2 almost-everywhere less than or equal to relation.

15. OTM $\triangleq$ Oracle Turing Machine.

16. $\mathrm{OTM_A} \triangleq$ Oracle Turing Machine under the answer-length-cost model.

17. $\mathrm{OTM_U} \triangleq$ Oracle Turing Machine under the unit-cost model.

18. $POTM \triangleq$ Polynomial-time computable Oracle Turing Machine.

19. **BFF** $\triangleq$ Basic Feasible Functional.

20. For $n \in \mathbf{N}$, $|n| \triangleq$ The length of the presentation of n.

21. For $f \in \mathcal{T}$, $|f| \triangleq$ The length function of f defined by

$$|f| = \lambda n \cdot \mathsf{max}(\{|f(x)| : |x| \leq n, x \in \mathsf{dom}(f)\}).$$

Note that f does not have to be a total function.

22. $\|\sigma\| = \sum\limits_{i \in \mathsf{dom}(\sigma)} (|i| + |\sigma(i)| + 2)$.

23. $\sigma^{\sim 0} \triangleq$ The zero extension of σ. That is, for every $x \in \mathbf{N}$,

$$\sigma^{\sim 0}(x) = \begin{cases} \sigma(x) & \text{if } x \in \mathsf{dom}(\sigma); \\ 0 & \text{otherwise.} \end{cases}$$

24. $\sigma^- \triangleq \sigma^- \subset \sigma$, and $\mathsf{dom}(\sigma^-) = \mathsf{dom}(\sigma) - \{\mathsf{max}(\mathsf{dom}(\sigma))\}$, where $\sigma \neq \emptyset$.

25. $f_{[n]} \triangleq$ The initial n-element segment of f, where f is a total function.

26. $\sigma_{[n]} \triangleq$ The initial n-element segment of σ, where $\mathsf{dom}(\sigma)$ is an initial segment of $\mathbf{N}$. By convention, if $\mathsf{max}(\mathsf{dom}(\sigma)) \leq n$, then $\sigma_{[n]} = \sigma$.

27. $(\!(\sigma, x)\!) = \left\{ (f, x) \;\middle|\; \sigma \subset f \right\}$.

28. $\varphi_e \triangleq$ The function computed by the Turing Machine with index e.

29. $\widehat{\varphi}_e \triangleq$ The functional computed by the Oracle Turing Machine with index e.

30. $\varphi_e(x) \downarrow^s \triangleq$ The Turing Machine with index e halts in s steps on input $x \in \mathbf{N}$.

31. $\widehat{\varphi}_e(f, x) \downarrow^s \triangleq$ The Oracle Turing Machine with index e halts in s steps on input $(f, x) \in \mathcal{T} \times \mathbf{N}$.

32. $\mathbf{T_2TB} \triangleq$ The set of Type-2 Time-Bounds of type $\mathcal{F} \times \mathbf{N} \to \mathbf{N}$. Unless stated otherwise, we let α, β, and γ range over $\mathbf{T_2TB}$.

33. $F_\beta \triangleq$ The limit functional determined by $\beta \in \mathbf{T_2TB}$.

34. $\widehat{\varphi}_{e,\beta} \triangleq$ The functional computed by the Oracle Turing Machine with index e clocked with $\beta \in \mathbf{T_2TB}$.

35. $\widehat{\varphi}_{e,\beta}(f, x) \Uparrow \triangleq$ The β-clocked Oracle Turing Machine with index e is clipped by the clock β during the course of computation on $(f, x) \in \mathcal{T} \times \mathbf{N}$.

36. $E_{e,\beta} \triangleq \left\{ (f, x) \;\middle|\; \widehat{\varphi}_{e,\beta}(f, x) \Uparrow \right\}$.

37. $\widehat{\Phi}_e(f, x) \triangleq$ The expenses needed to compute $\widehat{\varphi}_e(f, x)$. We use step-counting as our complexity measure.

38. $\widetilde{\Phi}_e(\sigma, x) : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$ is defined by

$$\widetilde{\Phi}_e(\sigma, x) = \begin{cases} \widehat{\Phi}_e(\sigma^{0\sim}, x), \text{ if } \widehat{\varphi}_e(\sigma^{0\sim}, x)\downarrow, \text{ and all said queries are in } \mathsf{dom}(\sigma); \\ \text{The expenses of the computation of } \widehat{\varphi}_e(\sigma^{0\sim}, x) \text{ right after the} \\ \qquad \text{first query outside } \mathsf{dom}(\sigma) \text{ is completed.} \end{cases}$$

[Note that $\widetilde{\Phi}_e$ is not necessarily recursive.]

39. $\langle\beta_i\rangle \triangleq$ A sequence of type-2 time bounds $\beta_0, \beta_1, \beta_2, \ldots$.

40. $\mathbf{C}(\beta) \triangleq$ The type-2 complexity class determined by $\beta \in \mathbf{T}_2\mathbf{TB}$.

41. $\mathbf{C}(\langle\beta_i\rangle) \triangleq \bigcup_i \mathbf{C}(\beta_i)$.

Bibliography

[AGM92] S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors. *Handbook of Logic in Computer Science*. Oxford University Press, 1992. Background: Mathematical Structures.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[Ang88] Dana Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[AS83] Dana Angluin and Carl Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15(3):237–269, September 1983.

[Bau64] John D. Baum. *Elements of Point Set Topology*. Prentic-Hall, 1964. First reprinted by Dover in 1991.

[BB75] Lenore Blum and Manuel Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.

[BC92] Stephen Bellantoni and Stephen Cook. A new recursive-theoretic characterization of the polytime functions. *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 97–110, 1992.

[Bel92] Stephen Bellantoni. Predicative recursion and computational complexity. Ph.d. dissertation, University of Toronto, 1992.

[BH74] Walter S. Brainerd and Landweber Lawrance H. *Theory of Computation*. John Wiley & Sons, New York, 1974.

[Blu67] Manuel Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM*, 14(2):322–336, 1967.

[Blu71] Manuel Blum. On effective procedures for speeding up algorithms. *Journal of the ACM*, 18(2):290–305, 1971.

[Bor69] 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.

[Bor72] A. Borodin. Computational complexity and the existence of complexity gaps. *Journal of the ACM*, 19(1):158–174, 1972.

[Bus86] S. Buss. The polynomial hierarchy and intuitionistic bounded arithmetic. In A. Selman, editor, *Structure in Complexity Theory: Lecture Notes in Computer Science*, volume 223, pages 77–103. Springer-Verlag, 1986.

[Cal88] Critian Calude. *Theories of Computational Complexity*. Number 35 in Annals of Discrete Mathematics. North-Holland, Elsevier Science Publisher, B.V., 1988.

[Cas86] John Case. Learning machines. In W. Demopoulos and A. Marras, editors, *Language Learning and Concept Acquisition*. Ablex Publishing Co., 1986.

[CG72] R. L. Constable and D. Gries. On classes of program schemata. *SIAM Journal of Computing*, 1.1:66–118, 1972.

[CK89] 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.

[CK90] Stephen A. Cook and Bruce M. Kapron. Characterization of the basic feasible functions of finite type. In S. Buss and P. Scott, editors, *Feasible Mathematics: A Mathematical Science Institute Workshop*, pages 71–95. Birkhäuser, 1990.

[CLR89] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, Massachusetts, 1989.

[Cob65] A. Cobham. The intrinsic computational difficulty of functions. *Proceedings of the International Conference on Logic, Methodology and Philosophy*, pages 24–30, 1965.

[Con72] Robert L. Constable. The operator gap. *Journal of the ACM*, 19:175–183, 1972.

[Con73] Robert L. Constable. Type two computational complexity. *Proceedings of the Fifth ACM Symposium on the Theory of Computing*, pages 108–122, 1973.

[Coo71] Stephen Cook. The complexity of theorem proving procedures. *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[Coo75] Stephen Cook. Feasible constructive proofs and the propositional calculus. *Proceedings of the 7th Annual ACM Symposium on the Theory of Computing*, pages 83–97, 1975.

[Coo91] Stephen A. Cook. Computability and complexity of higher type functions. In Y. N. Mpschovakis, editor, *Logic from Computer Science*, pages 51–72. Springer-Verlag, 1991.

[CS83] John Case and Carl Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.

[CU89] 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.

[Cut80] Nigel Cutland. *Computability: An introduction to recursive function theory.* Cambridge, New York, 1980.

[Dav58] Martin Davis. *Computability and Unsolvability.* McGraw-Hill, 1958. First reprinted by Dover in 1982.

[Dav65] Martin Davis, editor. *The Undecidable.* Raven Press, New York, 1965.

[DS86] Robert P. Daley and Carl Smith. On the complexity of inductive inference. *Information and Control*, 69:12–40, 1986.

[FR59] R. M. Friedberg and H. Rogers, Jr. Reducibility and completeness of sets of integers. *Z. Math. Logik Grundlag. Math.*, 5:117–125, 1959.

[GH77] R.O. Gandy and J.M.E. Hyland. Computable and recursively countable functions of higher type. *Logic Colloquium*, 76:405–438, 1977.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to Theory of NP-completeness.* Freeman, New York, 1979.

[Göd36] Kurt Gödel. Über die länge der beweise. *Ergebnisse eines Math. Kolloquiums*, 7:23–24, 1936. Translated and collected in [Dav65], pages 82-83, "On The Length of Proofs.".

[Gol67] E. Mark Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[GS92] William I. Gasarch and Carl H. Smith. Learning via queries. *Journal of the ACM*, 39(3):649–674, 1992.

[HH71] J. Hartmanis and J.E. Hopcroft. An overview of the theory of computational complexity. *Journal of the ACM*, 18:444–475, 1971.

[HS65] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transitions of the American Mathematics Society*, pages 285–306, May 1965.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Language, and Computation.* Addison-Wesley, 1979.

[IKR01a] Robert J. Irwin, Bruce M. Kapron, and James S. Royer. On characterizations of the basic feasible functionals: Part I. *Journal of Functional Programming*, 2001. (to appear).

[IKR01b] Robert J. Irwin, Bruce M. Kapron, and James S. Royer. On characterizations of the basic feasible functionals: Part II. *To appear*, pages ?–?, 2001?

[JORS99] Sanjay Jain, Daniel Osherson, James S. Royer, and Arun Sharma. *Systems That Learn*. The MIT Press, Cambridge, Massachusetts, second edition, 1999.

[JS94] Sanjay Jain and Arun Sharma. On the intrinsic complexity of language identification. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 278–286, 1994.

[Kap91] Bruce M. Kapron. Feasible computation in higher types. Ph.d. dissertation, University of Toronto, 1991.

[Kar72] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Tatcher, editors, *Complexity of Computer Computations*, pages 85–103. Prenum Press, 1972.

[KC91] 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.

[KC96] Bruce M. Kapron and Stephen A. Cook. A new characterization of type 2 feasibility. *SIAM Journal on Computing*, 25:117–132, 1996.

[Kle59] Steve C. Kleene. Recursive functionals and quantifies of finite types I. *Transitions of the American Mathematics Society*, 91:1–52, 1959.

[Kle60] Steve C. Kleene. Turing-machine computable functionals of finite types I. *Proceedings of the 1960 Congress for Logic, Methodology, and the Philosophy of Science*, pages 38–45, 1960.

[Kle62] Steve C. Kleene. Turing-machine computable functionals of finite types II. *Proceedings of London Mathematical Society*, 12:245–258, 1962.

[Kle63] Steve C. Kleene. Recursive functionals and quantifies of finite types II. *Transitions of the American Mathematics Society*, 108:106–142, 1963.

[Lei95] Daniel Leivant. Remified recurrence and computational complexity I: Word recurrence and poly-time. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1995.

[LR70] L.H. Landweber and E.R. Robertson. Recursive properties of abstract complexity classes. *ACM Symposium on the Theory of Complexity*, May 1970.

[Meh74] Kurt Mehlhorn. Polynomial and abstract subrecursive classes. *Proceedings of the Sixth Annual ACM Symposium on the Theory of Computing*, pages 96–109, 1974.

[Meh76] Kurt Mehlhorn. Polynominal and abstract subrecursive classes. *Journal of Computer and System Science*, 12:147–178, 1976.

[MF72] A. R. Meyer and P. C. Fischer. Computational speed-up by effective operators. *The Journal of Symbolic Logic*, 37:55–68, 1972.

[MM69] 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.

[Ner57] A. Nerode. General topology and partial recursive functionals. *Talks Cornell Summ. Inst. Symb. Log., Cornell*, pages 247–251, 1957.

[Nor80] Dag Normann. *Recursive on the Countable Functionals*, volume 811 of *Lecture Notes in Mathematics*. Springer-Verlag, New York, 1980.

[Odi89] Piergiorgio Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science Publishing, North-Holland, Amsterdam, 1989.

[Odi99] 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.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, New York, 1994.

[PE60] Marian B. Poúr-El. A comparison of five "computable" operators. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:325–340, 1960.

[Pez98] Elena Pezzoli. On the computational complexity of type two functionals, Proceedings of the Conference for Computer Science Logic '97. In D. van Dalen and M Mezem, editors, *Lecture Notes in Computer Science*, volume 1414, pages 373–388. Springer-Verlag, 1998.

[Rab60] M.O. Rabin. Degree of difficulty of computing a function and a partial ordering of recursive sets. Technical Report 2, Hebrew University, 1960.

[RC94] James Royer and John Case. *Subrecursive Programming Systems: Complexity & Succinctness*. Birkhäuser, 1994.

[Rog92] Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. The MIT Press, third edition, 1992. Original edition published by

McGraw-Hill in 1967. First paperback edition published by MIT Press in 1987.

[Roy97] 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.

[Sei90] Joel I. Seiferas. Machine-independent complexity theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 163–186. North-Holland, Elsevier Science Publisher, B.V., 1990. MIT press for paperback edition.

[Set92] Anil Seth. There is no recursive axiomatization for feasible functionals of type-2. *The Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 286–295, 1992.

[Set93] Anil Seth. Some desirable conditions for feasible functionals of type-2. *The Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 320–331, 1993.

[Set94] Anil Seth. Complexity theory of higher type functionals. Ph.d. dissertation, University of Bombay, 1994.

[Set95] Anil Seth. Turing machine characterizations of feasible functionals of all finite types. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 407–428. Birkhäuser, 1995.

[Sho95] J.R. Shoenfield. The mathematical works of S.C. Kleene. *The Bulletin of Symbolic Logic*, 1(1):9–43, March 1995.

[Soa87] Robert I. Soare. *Recursively Enumerable Sets and Degrees.* Perspectives in Mathematical Logic. Springer-Verlag, New York, 1987.

[Sym71] 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.

[Tow90] Mike Townsend. Complexity of type-2 relations. *Notre Dame Journal of Formal Logic*, 31(2):241–262, Spring 1990.

[Tur39] Alan M. Turing. System of logic based on ordinals. *Proceedings of the London Mathematical Society*, 45:161–228, 1939.

[Usp55] V.A. Uspenskii. On countable operations (Russian). *Doklady Akademii Nauk SSSR*, 103:773–776, 1955.

[Van75] P. Van Emde Boas. Ten years of speed-up. *Proceedings of the Fourth Symposium Mathematical Foundations of Computer Science 1975*, pages 13–29, 1975. Lecture Notes in Computer Science.

[WW85] Klaus Wagner and Gerd Wechsung. *Computational Complexity.* Mathematics and its applications (East European Series). D. Reidel Publishing Company, Dordrecht, 1985.

[You73] Paul Young. Easy construction in complexity theory: Gap and speed-up theorems. *Proceedings of the American Mathematical Society*, 37(2):555–563, February 1973.

Biographical Data

Name:	Chung-Chih Li
Date and Place of Birth:	November 11, 1962 Tainan, Taiwan
College:	Tamkang University Taipei, Taiwan B.S., Computer Science, 1986
Graduate:	Lamar University Beaumont, Texas, USA M.S., Computer Science, 1991
Graduate Work:	Syracuse University Syracuse, New York Teaching Assistant: 1995-2001 Research Assistant: 1999 ETS, New Jersey Research Fellow: 1995