On Type-2 Complexity Classes

Preliminary Report

Chung-Chih Li* James S. Royer*

15 March 2001

Abstract

There are now a number of things called "higher-type complexity classes." The most promenade of these is the class of *basic feasible functionals* [CU93, CK90], a fairly conservative higher-type analogue the (type-1) polynomial-time computable functions. There is however currently no satisfactory general notion of what a higher-type complexity class should be. In this paper we propose one such notion for type-2 functionals and begin an investigation of its properties. The most striking difference between our type-2 complexity classes and their type-1 counterparts is that, because of topological constrains, the type-2 classes have a much more ridged structure. *Example:* It follows from McCreight and Meyer's Union Theorem [MM69] that the (type-1) polynomial-time computable functions form a complexity class (in the strict sense of Definition 1 below). The analogous result *fails* for the class of type-2 basic feasible functionals.

§1. Introduction

Constable [Con73] was one of the first to study the computational complexity of higher-type functionals. In that 1973 paper, he raised two good questions:

1. What is the type-2 analogue of the polynomial-time computable functions?

2. What is the computational complexity theory of the type-2 effectively continuous functionals?

In the years since there has been a fair amount of attention given to addressing the first question, but hardly any to the second. We think that after nearly three decades the man deserves an answer. Herein we make a start at providing one.

*Dept. of Elec. Eng. & Computer Science; Syracuse University; Syracuse, NY 13244 USA.
Research supported in part by NSF grant CCR-9522987.

Professor Constable will have to wait a bit longer for a full answer to his question because what he seems to have had in mind in 1973 and what we study below are different in a couple of respects. First, Constable was interested in effectively continuous functionals [Odi89] of the general type $(\mathrm{N} \rightharpoonup \mathrm{N})^m \times \mathrm{N}^n \rightharpoonup \mathrm{N}$.[1] We instead focus on partial recursive functionals [Odi89] of type $(\mathrm{N} \to \mathrm{N}) \times \mathrm{N} \rightharpoonup \mathrm{N}$ — a much more tractable setting. Second, Constable wanted an extension of Blum's complexity measure axioms [Blu67, Odi99] to type-2 and included an interesting proposal along those lines. At present type-2 computational complexity is at so poorly understood that we believe that concrete, worked-out examples are what is needed. So instead of trying to develop an axiomatic treatment, we follow an approach similar to that of Hartmanis and Stearns [HS65] in studying the complexity properties of a simple, standard model of computation. In our case, the model is the deterministic, multi-tape, oracle Turing machine with Kapron and Cook's answer-length cost convention (more on this shortly).

Outline. The next section sketches a few facts about type-1 complexity theory and explains our focus on complexity classes. Section 3 introduces our model of type-2 computation and some associated notions. Section 4 considers the nature of type-2 time bounds and Section 5 concerns what it means for a type-2 time bound to hold almost everywhere. Section 6 introduces our definition of a type-2 complexity class and presents few elementary results about these classes. Unions of complexity classes and whether these unions are themselves complexity classes are considered in Section 7. Section 8 studies the problem of whether there is a uniformly way to expand a given complexity class to a strictly larger class. Finally, Section 9 contains our conclusions and suggestions for future work.

§2. *A glance at type-1 computational complexity*

Our study of type-2 computational complexity proceeds by rough analogy with the type-1 theory. Thus before considering the situation at type-2, we start by recalling a few basic facts about the type-1 theory.

Our model of computation. Following hoary complexity-theoretic tradition, we take deterministic, multi-tape Turing machines (TMs) as our default model of type-1 computation. Each step of a TM has unit cost. To simplify matters a bit, we also follow the common convention of requiring that each TM must read its entire input string. This forces a TM to have distinct computations on distinct inputs. (We will return to this point later.)

Strings and numbers. Each $x \in \mathrm{N}$ is identified with its dyadic representation over $\{\mathbf{0}, \mathbf{1}\}$. Thus, $0 \equiv \epsilon$, $1 \equiv \mathbf{0}$, $2 \equiv \mathbf{1}$, $3 \equiv \mathbf{00}$, etc. For each $x \in \mathrm{N}$,

[1] *Notation:* N denotes the set of natural numbers and $A \rightharpoonup B$ (respectively, $A \to B$) denotes the collection of all partial (respectively, total) set-theoretic functions from A to B.

$|x|$ denotes the length of its dyadic representation. We will freely pun between $x \in \mathrm{N}$ as a number and a **0**-**1**-string. TMs are thought of computing partial functions over N ($\cong \{\,\mathbf{0},\mathbf{1}\,\}^*$).

The standard indexing and complexity measure. $\mathcal{PR}$ and $\mathcal{R}$ respectively denote the partial recursive and total recursive functions over N. Let $\langle\varphi_i\rangle_{i\in\mathrm{N}}$ be an acceptable indexing [Rog67] of $\mathcal{PR}$ based on TMs. We call i a φ-program for φ_i. For each i and x, let $\Phi_i(x)$ denote the run time of the TM encoded by i on input x. Note that $\langle\Phi_i\rangle_{i\in\mathrm{N}}$ satisfies Blum's complexity measure axioms: (i) $\{\,(i,x) \mid \varphi_i(x)\!\downarrow\,\} = \{\,(i,x) \mid \Phi_i(x)\!\downarrow\,\}$ and (ii) $\{\,(i,x,n) \mid \Phi_i(x) \leq n\,\}$ is decidable. Also note that it follows from our requirement that a TM must read all of its input that $|x|+1 \leq \Phi_i(x)$ for each i and x.

Ordering on functions and almost everywhere relations. For $f,g\colon A \to B$, $f \leq g$ means that for all $x \in A$, $f(x) \leq g(x)$; $f < g$, and so on, are defined analogously. For $f,g\colon \mathrm{N} \to \mathrm{N}$, $f =^* g$ means that $\{\,x \mid f(x) = g(x)\,\}$ is co-finite; $f <^* g$, and so on, are defined analogously.

Definition 1 (Type-1 complexity classes). For each $t \in \mathcal{R}$:

$$C(t) \;=_{\mathrm{def}}\; \{\,\varphi_i \in \mathcal{R} \mid i \in \mathrm{N} \;\&\; \Phi_i \leq^* t\,\}. \tag{1}$$

We call $C(t)$ the *complexity class* named by t. ◇

Equation (1) is the standard definition of a complexity class relative to an arbitrary complexity measure Φ. By using special properties of our particular choice of Φ, we could replace the $\leq^*$ in (1) with $\leq$ and definite essentially the same notion. However, we retain the $\leq^*$ as a pedagogical reminder that membership in a complexity class depends on the *asymptotic* behavior of witnessing programs. For example, under many models of type-1 computation one can effectively patch programs so that on some specified finite set of arguments, the complexity is essentially anything you choose and off that finite set of arguments, the complexity is unchanged from the original. Thus, the "inherent complexity" of a program is only revealed in its asymptotic behavior.[2] One consequence of (1) is that to establish $f \notin C(t)$ for given f and t, one must prove that any program for f must have complexity that is infinitely often greater than t. Here is a sample argument along these lines.

Theorem 2 (Rabin [Rab60]). *Suppose $t \in \mathcal{R}$. Then there is an 0–1-valued element $f \in \mathcal{R}$ such that $f \notin C(t)$.*

Proof Sketch. The proof uses a standard cancellation constructions. In the program for f given in Figure 1, C_w = programs cancelled on inputs $< w$ and

[2] Another reason stems from recursive relatedness [Blu67, Odi99]; when you abstract away from a particular model of computation, the almost everywhere bounds are a necessary part of most theorems in the general theory.

Input x.

$C_0 \leftarrow \emptyset$.

For $w \leftarrow 0$ to x do:

$S_w \leftarrow \{\, k \leq w \mid k \notin C_w \ \ \& \ \ \Phi_k(w) \leq t(w) \,\}$.

If $S_w \neq \emptyset$ then $C_{w+1} \leftarrow C_w \cup \{\, \min(S_w) \,\}$ else $C_{w+1} \leftarrow C_w$.

If $S_x = \emptyset$ then return 0 else return $1 \dot{-} \varphi_e(x)$, where $e = \min(S_x)$.

Figure 1: The program for f

S_w = the candidates for cancellation on input w. A program i is *cancelled* on input w if and only if $i \in C_{w+1} - C_w$, in which case the construction will guarantee that $\Phi_i(w) \leq t(w)$, $f(w) \neq \varphi_i(w)$, and i will never be cancelled again.

It is clear from the program that cancellation works as advertised and f is a 0-1 element of $\mathcal{R}$. It remains to show $f \notin C(t)$. Suppose that i is such that $\Phi_i \leq^* t$. Choose $w_0 \geq i$ so that (a) $\Phi_i(w_0) \leq t(w_0)$ and (b) for each $k < i$ that is ever cancelled, k has been cancelled before input w_0. Hence, either i has been cancelled on a $w < w_0$ or the construction must cancel i on input w_0. In either case $\varphi_i \neq f$. Therefore, $f \notin C(t)$. □

Honesty. Note that Definition 1 says nothing about the complexity of computing t itself. This is an important issue that is usually dealt with through the notions of honesty and time constructibility.

Definition 3. Suppose $f, g\colon \mathrm{N} \to \mathrm{N}$.

(a) We say that f is *g-honest* if and only if for some i, a φ-program for f, $\Phi_i \leq g \circ f$.

(b) We say that f is *honest* if and only if f is g-honest for some $g \in \mathcal{R}$.

(c) We say that f is *time constructible* if and only if $f = \Phi_i$ for some i with $\varphi_i \in \mathcal{R}$. ◇

Intuitively, f is honest provided there is a way of computing f such that the size of each output is roughly commensurate with the time it takes to produce that output. The construction for Rabin's Theorem produces highly dishonest functions. On the other hand, a time constructible function is as honest as it is possible to be. Roughly, honest functions provide good names for complexity classes, whereas complexity classes named by dishonest functions can be quite pathological (see Theorems 23 and 24 below). In this paper we will not deal directly with type-2 analogues of honesty and time constructibility. However, they will be important background concerns, and we will see that there is some amount of honesty built into our notion of type-2 time bound.

Why are complexity classes of interest? One of the central obsessions of computation complexity theory is the attempt to draw sharp boundaries between the computationally feasible and infeasible, where the notions of feasible and infeasible vary with context. Given an arbitrary $t \in \mathcal{R}$, $C(t)$ is unlikely to represent anyone's notion of feasibility. However, $C(t)$ is a very simple and elegant way of representing a complexity-theoretic boundary. Thus, if you want to understand computational complexity, one of the first things you want to study is the nature of these boundaries.

This is enough about the type-1 theory for the moment. Our goal in the next few sections is to introduce a sensible type-2 analogue of Definition 1.

§3. Type-2 computations and their costs

For our default model of type-2 computation we take deterministic, multi-tape *oracle Turing machines* (OTMs). Under our setup OTMs are TMs that are augmented with two special tapes: a *query tape* and a *reply tape*, and one special instruction: *query*. To make a query of an oracle $f\colon \mathrm{N} \to \mathrm{N}$, an OTM writes a **0**-**1** string (interpreted as the dyadic representation of an $x \in \mathrm{N}$) on the query tape and goes into its query state, whereupon the contents of the query tape are erased and the contents of the reply tape become the dyadic representation of $f(x)$. We also require that each OTM must read all of its type-0 input before halting, and additionally, that immediately after making the query, an OTM must read all of the answer to said query. Each step of an OTM has unit cost, but our requirement that OTMs read all of each oracle response makes our cost model equivalent to Kapron and Cook's *answer-length cost model* [KC96].

Why OTMs? No one outside of complexity theory cares much for TMs or OTMs as models of computation. So our use of OTMs is a poor marketing choice. In their favor, OTMs are a simple, conservative model of computation with a simple, conservation notion of cost. Hence, reasoning about their complexity is straightforward (or as straightforward as reasoning about complexity can ever be) and this just what we want from our model of computation in our initial foray into this territory. Extending our results to other basic models of computation should not be that hard, but we need to know the general shape of the results first.

The unit cost model for OTMs. If we drop the requirement that each OTM must read the entire answer to each query, then we obtain the *unit cost model* for OTMs. Working with this model is more difficult than the answer-length cost model and the results tend to be weaker. This unit cost model is studied in [Li01], but we shall not discuss it further in this paper.

Finite functions. Let $\mathcal{F}$ denote the collection of finite functions over N, i.e., each $\sigma\colon \mathrm{N} \rightharpoonup \mathrm{N}$ is defined on only finitely many arguments. In the following σ

and τ (with or without decorations) range over $\mathcal{F}$. We identify each σ with its graph: $\{\,(x,\sigma(x)) \mid \sigma(x)\downarrow\,\}$. We shall assume some canonical representation of the elements of $\mathcal{F}$ and typically treat them as type-0 arguments to functions. For each σ, define $\overline{\sigma}\colon \mathrm{N} \to \mathrm{N}$ by:

$$\overline{\sigma}(x) \;=\; \begin{cases} \sigma(x), & \text{if } \sigma(x)\downarrow; \\ 0, & \text{otherwise.} \end{cases}$$

The standard indexing and complexity measure. The class of functionals computed the OTMs sketched above are called the *partial recursive functionals* (of type $(\mathrm{N} \to \mathrm{N}) \times \mathrm{N} \rightharpoonup \mathrm{N}$) in Odifreddi [Odi89]. We denote this class by $\mathcal{PRF}$ and the total members of this class by $\mathcal{RF}$. Let $\langle \boldsymbol{\varphi}_i \rangle_{i\in\mathrm{N}}$ be an acceptable indexing of $\mathcal{PRF}$ based on OTMs. We call i a $\boldsymbol{\varphi}$-program for $\boldsymbol{\varphi}_i$. For each i, f, and x, let $\boldsymbol{\Phi}_i(f,x)$ denote the run time of the OTM encoded by i on input (f,x). Note that it follows from our requirement that an OTM must read all of its input that $|x|+1 \leq \boldsymbol{\Phi}_i(f,x)$ for each i, f, and x. For each i, f, x, and n, define $Q_i(f,x,n) =$ the set of queries issued in the first $\min(n, \boldsymbol{\Phi}_i(f,x))$ steps of the computation of $\boldsymbol{\varphi}$-program i on input (f,x), $Q_i(f,x) = \cup_n Q_i(f,x,n)$, $\mathrm{Use}_i(f,x,n) = \{\,(x,f(x)) : x \in Q_i(f,x,n)\,\}$, and $\mathrm{Use}_i(f,x) = \cup_n \mathrm{Use}_i(f,x,n)$. Also, for each i, σ, and x, we define

$$\boldsymbol{\Phi}_i(\sigma,x) \;=\; \begin{cases} \boldsymbol{\Phi}_i(\overline{\sigma},x), & \text{if } Q_i(\overline{\sigma},x) \subseteq \{\,y \mid \sigma(y)\downarrow\,\}; \\ n, & \text{otherwise, where } n \text{ is the number of steps} \\ & \quad \text{taken up to the issuance of the first query} \\ & \quad \text{“}\sigma(y) =?\text{”} \text{ where } \sigma(y)\uparrow. \end{cases}$$

§4. *Type-2 time bounds*

Our current goal is to lift Definition 1 to type-level 2 in a reasonable way. The key issue in this is what should be the type-2 translation of the inequality $\Phi_i \leq^* t$ of (1). In place of Φ_i we clearly should use $\boldsymbol{\Phi}_i$, but there are two harder questions:

1. What should stand in place of $\leq^*$?
2. What should stand in place of t?

We examine the first question in the next section. Here we shall consider how to sensibly express time bounds on type-2 computations.

What we don't do, and why. One way to proceed is to use arbitrary elements of $\mathcal{RF}$ as time bounds. That is, given $T \in \mathcal{RF}$, we could say that $\boldsymbol{\varphi}$-program i has complexity everywhere bounded by T if and only if $\boldsymbol{\Phi}_i \leq T$ (i.e., for all f and x, $\boldsymbol{\Phi}_i(f,x) \leq T(f,x)$). Something of this sort is briefly considered by Kapron [Kap91] and Seth [Set94]. This sort of bound has the following troublesome feature.

PROPOSITION 4. *Suppose that $\mathbf{\Phi}_i \leq T$ and that b is some $\boldsymbol{\varphi}$-program b for T. Then, for all f and x, $Q_i(f,x) \subseteq Q_b(f,x)$.*

PROOF. Suppose by way of contradiction that $y \in (Q_i(f,x) - Q_b(f,x))$ for some particular f and x. Let f' be such that $f'(z) = f(z)$ for $z \neq y$ and $f'(y) = 2^{1+T(f,x)}$. Then $T(f',x) = T(f,x)$ since $\boldsymbol{\varphi}$-program b fails to query f on y. Moreover, $\mathbf{\Phi}_i(f',x) > T(f,x)$ since $\boldsymbol{\varphi}$-program i on input (f',x) will query f' on y and the cost of this query is greater than $T(f,x)$. Therefore, $\mathbf{\Phi}_i(f',x) > T(f,x) = T(f',x)$, a contradiction. □

Thus, for $\mathbf{\Phi}_i \leq T$ to hold it must be the case that for any $\boldsymbol{\varphi}$-program b for T and any input (f,x), the $\boldsymbol{\varphi}$-program b must anticipate all of the possible questions the computation of i on (f,x) might ask and ask them itself. This seems like an odd thing for a humble bound to do. In particular, if T is honest and small (in some reasonable senses), then $\{\, \boldsymbol{\varphi}_i \in \mathcal{RF} \mid i \in \mathrm{N} \;\&\; \mathbf{\Phi}_i \leq T \,\}$ must be very restricted since the set of queries a $\boldsymbol{\varphi}_i$ in this collection will be quite circumscribed.

Our approach. To avoid having our bounding functionals issue queries, we make them a particular sort of enumeration operator [Rog67, Odi89]. That is, the bounding functionals can be thought of as passive observers of the computations they are set to bound; at any point of the computation, the bounding functional will have "bounding value" based on what the functional has seen of the input and the queries. This scheme is directly inspired by the standard clocking scheme for second-order polynomially bounded OTMs [KC96, Set92]. We proceed formally as follows.

DEFINITION 5. Suppose $\beta\colon \mathcal{F} \times \mathrm{N} \to \mathrm{N}$ is total computable.

(a) We say that β determines a *weak type-2 time bound* if and only if it satisfies the following three conditions, for all f, σ, and x,

Nontriviality: $\beta(\sigma,x) \geq |x| + 1$.
Convergence: $\lim_{\tau \to f} \beta(\tau,x)\!\downarrow < \infty$.
Boundedness: $\sup_{\tau \subset f} \beta(\tau,x) = \lim_{\tau \to f} \beta(\tau,x)$.

Let WB be the collection of all such β's.

(b) We say that β determines a *strong type-2 time bound* if and only if β satisfies the nontriviality and convergence conditions as above as well as satisfying, for all σ, σ', and x:

Monotonicity: $\sigma \subseteq \sigma'$ implies $\beta(\sigma,x) \leq \beta(\sigma',x)$.

Let SB be the collection of all such β's. ◇

Clearly, if SB $\subset$ WB. Unless we say otherwise, β will denote an element of WB in the following.

Definition 6.

(a) We say that the run time of $\boldsymbol{\varphi}$-program i on input (f, x) is *bounded* by β (written $\boldsymbol{\varphi}_{i,\beta}(f, x)\Downarrow$) if and only if, for each n, $\Phi(i, \sigma_n, x) \leq \beta(\sigma_n, x)$, where $\sigma_n = \mathrm{Use}_i(f, x, n)$.

(b) We say that the computation of $\boldsymbol{\varphi}$-program i on input (f, x) is *clipped* by β (written $\boldsymbol{\varphi}_{i,\beta}(f, x)\Uparrow$) if and only if not $\boldsymbol{\varphi}_{i,\beta}(f, x)\Downarrow$.

(c) Define $E_{i,\beta} = \{\, (f, x) \mid \boldsymbol{\varphi}_{i,\beta}(f, x)\Uparrow \,\}$; we call $E_{i,\beta}$ the *exception set* for i and β.

(d) We say that the run time of $\boldsymbol{\varphi}$-program i is *everywhere bounded* by β if and only if $E_{i,\beta}$ is empty. $\diamond$

Example 7.

(a) Suppose $\boldsymbol{\varphi}_i \in \mathcal{RF}$ and, for each σ and x, let $\beta(\sigma, x) = \boldsymbol{\Phi}_i(\sigma, x)$. Then $\beta \in \mathrm{SB}$ and it is no surprise that the run time of $\boldsymbol{\varphi}$-program i is everywhere bounded by β.

(b) For each a, k, d, x, and σ, define $\beta_{a,k,0}(\sigma, x) = a \cdot (|x| + 1)^k$ and also $\beta_{a,k,d+1}(\sigma, x) = a \cdot (|w| + |x| + 1)^k$, where $w = \max(\{\, \sigma(y) \mid |y| \leq \beta_{a,k,d}(\sigma, x) \;\;\&\;\; \sigma(y)\downarrow \,\})$. Then each $\beta_{a,k,d} \in \mathrm{SB}$ and the class of BFFs of type $(\mathrm{N} \to \mathrm{N}) \times \mathrm{N} \to \mathrm{N}$ is exactly $\bigcup_{a,k,d} \{\, \boldsymbol{\varphi}_i \mid$ the run time of $\boldsymbol{\varphi}$-program i is everywhere bounded by $\beta_{a,k,d} \,\}$ [Set92, IKR01]. $\diamond$

§5. *Type-2 almost everywhere bounds*

We want to speak of the run time of $\boldsymbol{\varphi}$-program i being *almost everywhere* bounded by β. Intuitively, this should mean that in some appropriate sense $E_{i,\beta}$ is finite. In the realm of function spaces, "finite" usually corresponds to compact in some topology. So the question of almost everywhere bounds comes down to a choice of topology.

What we don't do, and why. Our OTMs compute over $(\mathrm{N} \to \mathrm{N}) \times \mathrm{N}$. That space is isomorphic to N^{N} which has a well-known topology due to Baire. Let $\mathcal{B}$ denote this topology on $(\mathrm{N} \to \mathrm{N}) \times \mathrm{N}$. For $\mathcal{B}$, it suffices to take $\{\, (\!|\sigma, x|\!) \mid \sigma \in \mathcal{F},\, x \in \mathrm{N} \,\}$ as the collection of basic open sets, where for each σ and x,

$$(\!|\sigma, x|\!) \;\; =_{\mathrm{def}} \;\; \{\, (f, x) \mid f \supset \sigma \,\}.$$

The problem with $\mathcal{B}$ is that the compact sets are all too small for our purposes. This is shown by:

Proposition 8. *If an $E_{i,\beta}$ is $\mathcal{B}$-compact, then this $E_{i,\beta}$ is empty.*

Proof. It follows from Definitions 5(a) and 6(c) that $E_{i,\beta}$ is open in $\mathcal{B}$. But the only open compact set in this topology is $\emptyset$. $\square$

Roughly, the more open sets one has in a topology, the more restricted are its compact sets. $\mathcal{B}$ and "large" topologies in general thus fail to provide

sufficiently large compact sets so as to obtain nontrivial almost everywhere relations.

Our approach. To address the problem of what topology to use, we shift our attention from the set of possible inputs of an OTM to the set of possible *computations* of an OTM. To motivate this shift, let us first consider a particular ordinary TM M that ignores the convention about reading its entire input. This M acts as follows:

> Upon staring, M examines the first symbol on the input tape. If this is a $\mathbf{0}$, M immediately halts with output $\mathbf{0}$; otherwise, M reads the rest of the input and then halts with output $\mathbf{1}$.

Clearly there are infinitely many inputs on which M halts with output $\mathbf{0}$. However, there is only one computation of M that produces output $\mathbf{0}$: on an input of the form $\mathbf{0}\{\,\mathbf{0},\mathbf{1}\,\}^*$ the machine never looks beyond the initial $\mathbf{0}$, hence all such inputs produce the same computation. Therefore if we want to say that such an M does something for all but finitely many cases, we must specify whether cases in question are inputs or computations — each choice has its own faults and merits.

Now, any halting computation of an OTM has $2^{\aleph_0}$-many inputs which produce that computation. So if we want to say that OTM does something for all but finitely many cases, we again must choose between these cases corresponding to inputs or computations. $\mathcal{B}$ roughly corresponds to the "inputs" choice. No single topology corresponds to the "computations" choice, but we need not be restricted to a single topology. The next two definitions introduce several topologies we need to consider.

Definition 9. Suppose $F\colon (\mathrm{N}\to\mathrm{N})\times\mathrm{N}\rightharpoonup\mathrm{N}$ is $\mathcal{B}$-continuous.

(a) A *locking segment* for F is a (σ,x) for which there is a $y\in\mathrm{N}$ such that for all f with $(f,x)\in(\!(\sigma,x)\!)$, $F(f,x)=y$.

(b) A *minimal locking segment* for F is a locking segment (σ,x) for F such that for each $\tau\subset\sigma$, (τ,x) fails to be a locking segment.

(c) The *induced topology* for F (denoted $\mathcal{I}(F)$) is the topology determined by the subbasis: $\{\,(\!(\sigma,x)\!)\mid(\sigma,x)$ is a minimal locking segment for $F\,\}$. ◇

Definition 10. The *induced topology* for the $\boldsymbol{\varphi}$-program i (denoted $\mathcal{I}_i$) is the topology determined by the subbasis: $\{\,(\!(\mathrm{Use}_i(f,x),x)\!)\mid f\colon(\mathrm{N}\to\mathrm{N})\to\mathrm{N},\ x\in\mathrm{N}\,\}$. ◇

It is easily seen, for each i with $\boldsymbol{\varphi}_i\in\mathcal{RF}$, that $\mathcal{I}(\boldsymbol{\varphi}_i)$ is a subtopology of $\mathcal{I}_i$ which in turn is a subtopology of $\mathcal{B}$ and that $\mathcal{I}(\boldsymbol{\varphi}_i)$ is *the* smallest subtopology of $\mathcal{B}$ such that $\boldsymbol{\varphi}_i$ is continuous.

Now we have a decision to make. We can take "the run time of $\boldsymbol{\varphi}$-program i is almost everywhere bounded by β" as meaning either (a) $E_{i,\beta}$ is $\mathcal{I}_i$-compact or (b) $E_{i,\beta}$ is $\mathcal{I}(\boldsymbol{\varphi}_i)$-compact. Choice (a) exactly matches our talk about counting computations. But working with choice (a) turns out to be tricky. Part of the problem is that under choice (a) it is very hard to compare computations of programs for the same functional — $\boldsymbol{\varphi}_i = \boldsymbol{\varphi}_j$ does not imply that $\mathcal{I}_i$ and $\mathcal{I}_j$ have much to do with one another. So for reason simplicity, in *this* paper we make choice (b). There are prices to be paid for this choice, but they are generally tolerable. Thus we officially introduce:

Definition 11. We say that the run time of $\boldsymbol{\varphi}$-program i is *almost everywhere bounded by* β if and only if $E_{i,\beta}$ is $\mathcal{I}(\boldsymbol{\varphi}_i)$-compact. $\diamond$

Note: Since $\mathcal{I}(\boldsymbol{\varphi}_i)$ is a subtopology of $\mathcal{I}_i$, any $\mathcal{I}_i$-compact set is also $\mathcal{I}(\boldsymbol{\varphi}_i)$-compact. We shall use this frequently in the following.

As a first check that Definition 11 is reasonable, we note:

Proposition 12. *Suppose i is such that $\boldsymbol{\varphi}_i \in \mathcal{RF}$ and $c \in \mathrm{N}$. Then $\{ (f,x) \mid \boldsymbol{\Phi}_i(f,x) \leq c \}$ is $\mathcal{I}(\boldsymbol{\varphi}_i)$-compact.*

Proof. Suppose that $\boldsymbol{\Phi}_i(f,x) \leq c$. It follows from our restrictions on OTMs that $|x|$, $|\max(Q_i(f,x))|$, and $|\max\{ \mathrm{Use}_i(f,x)(y) \mid y \in Q_i(f,x) \}|$ are all no greater than c. Clearly then, there are only finitely-many computations of $\boldsymbol{\varphi}$-program i with $\boldsymbol{\Phi}_i(f,x) \leq c$. Therefore, $\{ (f,x) \mid \boldsymbol{\Phi}_i(f,x) \leq c \}$ is $\mathcal{I}_i$-compact, and hence, $\mathcal{I}(\boldsymbol{\varphi}_i)$-compact. $\square$

§6. Type-2 complexity classes

Now that all the pieces are in place, we can state:

Definition 13. For each $\beta \in \mathrm{WB}$:

$$\mathbf{C}(\beta) \quad =_{\mathrm{def}} \quad \{ \boldsymbol{\varphi}_i \in \mathcal{RF} \mid i \in \mathrm{N} \ \&\ E_{i,\beta} \text{ is } \mathcal{I}(\boldsymbol{\varphi}_i)\text{-compact} \}. \tag{2}$$

We call $\mathbf{C}(\beta)$ the *complexity class* named by β. $\diamond$

This notion of complexity class is similar its type-1 cousin in many ways. Here is a first illustration.

Proposition 14. *Suppose $F \in \mathbf{C}(\beta)$. Then there is a $\boldsymbol{\varphi}$-program i for F and a $c \in \mathrm{N}$ such that $E_{i,c\cdot\beta} = \emptyset$.*

PROOF SKETCH. Let p be such that $\boldsymbol{\varphi}_p = F$ and $E_{p,\beta}$ is $\mathcal{I}(F)$-compact. Let $\mathbf{M}$ be the OTM coded by p and let $\mathcal{C}$ be a finite $\mathcal{I}(F)$-cover of $E_{p,\beta}$. If $\mathcal{C} = \emptyset$, then we are done. Suppose $\mathcal{C} \neq \emptyset$ and let $\{ x_0, \ldots, x_k \} = \{ x \mid \langle\!\langle \sigma, x \rangle\!\rangle \in \mathcal{C} \}$. One can argue that, for each $i \leq k$, there is a finite decision tree T_i as follows.

Each node n of T_i is labeled a $y_n \in \mathrm{N}$. If n is an interior node, this will correspond to the oracle query "$f(y_n) =?$". If n is a terminal node, this will correspond to the output being y_n. Each edge leaving an interior node is labeled a $z \in \mathrm{N}$; this corresponds to z being the answer to the interior node's query. For each node n of T_i, let σ_n be the finite function that corresponds to the set of queries and answers on the path leading to n. We require that (i) if n is a terminal node, then (σ_n, x_i) a locking segment for $\boldsymbol{\varphi}_p$ and $\boldsymbol{\varphi}_i(\overline{\sigma_n}, x_i) = y_n$; (ii) if n is an interior node, then for all $f \supset \sigma_n$, $\mathbf{M}$ on (f, x_i) queries f on y_n, and (iii) $\{ \langle\!\langle \sigma_n, x_i \rangle\!\rangle \mid n$ is a terminal node of T_i and $i \leq k \}$ covers $E_{p,\beta}$.

Given these T_i's, let $\mathbf{M}'$ be the OTM that, on input (f, x), checks if $x = x_i$ for some $i \leq k$. If not, then $\mathbf{M}'$ acts like $\mathbf{M}$. If so, then $\mathbf{M}'$ follows the decision tree T_i until either (i) it reaches an terminal node n, in which case is outputs y_n and halts or else (ii) $\mathbf{M}'$ reaches an interior node n, and $f(y_n)$ is not the label of any edge leaving n, in which case, $\mathbf{M}'$ acts like $\mathbf{M}$ on input (f, x).

Clearly, $\mathbf{M}'$ computes $\boldsymbol{\varphi}_p$. The extra cost of running $\mathbf{M}'$ on (f, x) over running $\mathbf{M}$ is the cost of following the decision tree T_i when $x = x_i$ for some $i \leq k$. Since in following the decision tree T_i simply involves making queries that $\mathbf{M}$ on input (f, x) will have to make anyhow. Hence, with a little careful programming, there is a $c \in \mathrm{N}$ such that $c \cdot \beta$ everywhere bounds the run time of $\mathbf{M}'$. □

Note: In general it is false that if $E_{i,\beta}$ is $\mathcal{I}(\boldsymbol{\varphi}_i)$-compact, then there is a c such that $E_{i,\beta+c} = \emptyset$ — this is part of the price of using the $\mathcal{I}(\boldsymbol{\varphi}_i)$ topology.

As a second illustration of the similarity between type-1 and type-2 complexity classes, we show that a straightforward lift of the proof of Rabin's Theorem (Theorem 2) suffices to obtain a type-2 version of that result.

THEOREM 15. *Suppose $\beta \in \mathrm{WB}$. Then there is an 0–1-valued element $F \in \mathcal{RF}$ such that $F \notin \mathbf{C}(\beta)$.*

PROOF SKETCH. The argument is a direct lift of the one given for Theorem 2 above. In the program for F given in Figure 2, $C_{f,w}$ = programs cancelled on inputs (f, w') with $w' < w$ and $S_{f,w}$ = the candidates for cancellation on input (f, w). A program i is *cancelled* on input (f, w) if and only if $i \in C_{f,w+1} - C_{f,w}$, in which case the construction will guarantee that $\boldsymbol{\varphi}_{i,\beta}(f, w)\Downarrow$, $F(f, w) \neq \boldsymbol{\varphi}_i(f, w)$, and i will never be cancelled again on an input of the form (f, x) with $x > w$.

It is clear from the program that cancellation works as advertised and F is a 0-1 element of $\mathcal{RF}$. To show $F \notin \mathbf{C}(\beta)$ consider an i such that $E_{i,\beta}$ is

Input (f, x).

$C_{f,0} \leftarrow \emptyset$.

For $w \leftarrow 0$ to x do:

$S_{f,w} \leftarrow \{\, k \leq w \mid k \notin C_{f,w} \;\&\; \boldsymbol{\varphi}_{k,\beta}(f, w)\Downarrow \,\}$.

If $S_{f,w} \neq \emptyset$ then $C_{f,w+1} \leftarrow C_{f,w} \cup \{\, \min(S_{f,w}) \,\}$ else $C_{f.w+1} \leftarrow C_{f,w}$.

If $S_{f,x} = \emptyset$ then return 0 else return $1 \dot{-} \boldsymbol{\varphi}_e(f, x)$, where $e = \min(S_{f,x})$.

Figure 2: The program for F

$\mathcal{I}(F)$-compact. Fix $f\colon \mathrm{N} \to \mathrm{N}$ and choose $w_0 \geq i$ so that (a) $\boldsymbol{\Phi}_{i,\beta}(f, w_0)\Downarrow$ and (b) for all $k < i$ that are ever cancelled on an input of the form (f, x) have been cancelled by input w_0. Hence, either i has been cancelled on a (f, w) with $w < w_0$ or the construction must cancel i on input (f, w_0). In either case $\boldsymbol{\varphi}_i \neq F$. Therefore, $F \notin C(\beta)$. □

So much for similarities, the next two sections demonstrate some marked differences between type-1 and type-2 complexity classes. To keep this paper a reasonable size we shall omit proofs in these next two sections, but the proofs can be found in [Li01].[3]

§7. *Unions of complexity classes*

The type-1 situation. The class of type-1 polynomial-time computable functions is commonly referred to as a complexity class, but it is far from obvious that there is a $t_P \in \mathcal{R}$ that names exactly that class. That there is such a t_P follows from the following quite difficult result, which holds when Φ is an arbitrary complexity measure.

Theorem 16 (The Union Theorem, McCreight and Meyer [MM69, Odi99]). *Suppose that $t\colon \mathrm{N}^2 \to \mathrm{N}$ is computable and nondecreasing in its first argument. Then there is a computable $g\colon \mathrm{N} \to \mathrm{N}$ such that $C(g) = \bigcup_i C(\lambda x.t(i, x))$.*

This theorem is barely true in the sense that you want the g of the theorem to have any nice properties (e.g., honesty), then you find the result breaks.

The type-2 situation. Since the Union Theorem is fairly delicate, it is no surprise that it is fails to hold in its full strength at type-2. However, the failure is spectacular. Here is an important example of this.

Theorem 17 (Li [Li01]). *The class of type-2 basic feasible functionals fails to be a type-2 complexity class.*

[3] A late draft of this is available as: ftp://ftp.cis.syr.edu/users/royer/CCLthesis.ps.

To obtain some measure of how bad this failure is and to obtain some sufficient conditions on a weak version the union theorem at type-2, we introduce some conditions on type-2 complexity bounds. To keep this paper a reasonable length, we shall not explain these notions beyond their definitions.

Definition 18.

(a) We say that (σ, x) is a *locking fragment* of β (denoted $\beta(\sigma, x)\downarrow$) if and only if for all $\tau \supseteq \sigma$, $\beta(\tau, x) = \beta(\sigma, x)$.

(b) We say that $\ell\colon \mathcal{F} \times \mathrm{N} \to \{0, 1\}$ is a *locking detector* for β if and only if (i) ℓ is computable, (ii) for each f and x, $\lim_{\sigma \to f} \ell(\sigma, x) = 1$, and (iii) for each σ and x, $\ell(\sigma, x) = 1$ implies that $\beta(\sigma, x)\downarrow$.

(c) A *minimal locking fragment* of β is a locking fragment of β such that, for all for all $\tau \subseteq \sigma$, (τ, x) fails to be a locking fragment of β.

(d) We say that β is *useful* if and only if for every (σ, x), minimal locking fragment of β, we have that, for each $\tau \subseteq \sigma$, $\beta(\tau, x) \geq \|\tau\| + |x| + 2$.[4] $\diamond$

Definition 19. Let $\langle \beta_i \rangle_{i \in \mathrm{N}}$ be a sequence of elements of WB such that the function $\lambda i, \sigma, x.\beta_i(\sigma, x)$ is computable. We say that:

(a) $\langle \beta_i \rangle_{i \in \mathrm{N}}$ is *ascending* if and only if, for all i, $\beta_i \leq \beta_{i+1}$.

(b) $\langle \beta_i \rangle_{i \in \mathrm{N}}$ is *useful* if and only if each β_i is useful.

(c) $\langle \beta_i \rangle_{i \in \mathrm{N}}$ is *convergent* if and only if, for each f and x, there is a $\sigma_{f,x} \subset f$ such that for all i, $\beta_i(\sigma_{f,x}, x)\downarrow$.

(d) $\langle \beta_i \rangle_{i \in \mathrm{N}}$ *uniformly convergent* if and only if, for all i, x, and σ, if $\beta_i(\sigma, x)\downarrow$, then for all j, $\beta_j(\sigma, x)\downarrow$.

(e) $\langle \beta_i \rangle_{i \in \mathrm{N}}$ *strongly convergent* if and only if $\langle \beta_i \rangle_{i \in \mathrm{N}}$ is uniformly convergent and there is a locking detector for β_0. $\diamond$

Theorem 20 (Li [Li01]). *There is a ascending, useful, convergent $\langle \beta_i \rangle_{i \in \mathrm{N}}$ such that $\bigcup_i \mathbf{C}(\beta_i)$ is not a complexity class.*

This is a fairly strong non-union result. We conjecture that convergent can be strengthened to uniformly convergent in the previous theorem. By strengthening the hypotheses on the β_i's even more, we can obtain the following weak union theorem. We conjecture that this theorem fails if we require all the complexity bounds to be elements of SB.

Theorem 21 (The Weak Type-2 Union Theorem, Li [Li01]). *Suppose that $\langle \beta_i \rangle_{i \in \mathrm{N}}$ is ascending, useful, and strongly convergent. Then there is a $\beta \in \mathrm{WB}$ such that $\mathbf{C}(\beta) = \bigcup_i \mathbf{C}(\beta_i)$.*

[4] There is a different definition of useful in [Li01] that is more understandable, but requires a bit of back-story.

One nice consequence of this theorem is that type-2 big-O classes *are* complexity classes. We conjecture, however, that the SB version of the following is false.

COROLLARY 22. *Suppose* $\beta \in \mathrm{WB}$. *Let* $\boldsymbol{O}(\beta) = \bigcup_{a,b\in\mathrm{N}} \mathbf{C}(a \cdot \beta + b)$. *Then* $\boldsymbol{O}(\beta)$ *is a complexity class.*

§8. Gaps and compressions

The type-1 situation. We know by Rabin's Theorem that for each $t \in \mathcal{R}$, there is a $t' \in \mathcal{R}$ such that $C(t) \subsetneq C(t')$. However, effectively constructing such a t' from a given t turns out to be impossible as shown by the following two theorems. (**N.B.** Constructing such a t' from a *program* for t is easy, but that is not the issue here.)

THEOREM 23 (THE GAP THEOREM, BORODIN [BOR72]). *For each* $r \in \mathcal{R}$, *there is an increasing* $t \in \mathcal{R}$ *such that* $C(t) = C(r \circ t)$, *in fact, there is no* i *with* $t \leq^* \Phi_i \leq^* r \circ t$.

THEOREM 24 (THE OPERATOR GAP THEOREM, CONSTABLE [CON72] AND YOUNG [YOU73]). *For each recursive operator* $\Theta\colon (\mathrm{N} \to \mathrm{N}) \to (\mathrm{N} \to \mathrm{N})$, *there is an increasing* $t \in \mathcal{R}$ *such that* $C(t) = C(\Theta(t))$, *in fact, there is no* i *with* $t \leq^* \Phi_i \leq^* \Theta(t)$.

In both of these theorems, the reason for the gap in which no Φ_i lives is that the t's in question are pathologically dishonest. If we restrict our attention to more sensible names for complexity classes, we obtain the following result that matches our intuitions a bit better.

THEOREM 25 (THE COMPRESSION THEOREM, BLUM [BLU67]). *There is a computable* $r\colon \mathrm{N}^2 \to \mathrm{N}$ *such that for all* i *with* $\varphi_i \in \mathcal{R}$, *we have* $C(\Phi_i) \subsetneq C(\lambda x.r(x, \Phi_i(x)))$.

The type-2 situation. The fact that the β's are tied to queries imposes a sort of honesty on our time bounds. We thus loose the gap phenomenon at type-2 as shown by:

THEOREM 26 (THE WB INFLATION THEOREM, LI [LI01]). *There is a recursive operator* Θ *such that, for each* $\beta \in \mathrm{WB}$, $\Theta(\beta) \in \mathrm{WB}$ *and* $\mathbf{C}(\beta) \subsetneq \mathbf{C}(\Theta(\beta))$.

We do obtain a gap theorem for unions. But given the wiggly nature of unions, this is not surprising nor is it particularly hard to show.

THEOREM 27 (THE UNION-GAP THEOREM, LI [LI01]). *For each recursive operator* Θ *such that for each* $\beta \in \mathrm{WB}$, $\Theta(\beta) \in \mathrm{WB}$, *there is an ascending* $\langle \beta_i \rangle_{i\in\mathrm{N}}$ *such that* $\bigcup_i \mathbf{C}(\beta_i) = \bigcup_i \mathbf{C}(\Theta(\beta_i))$.

§9. Conclusion

General type-2 complexity theory is almost completely unknown territory. In this paper we have blazed one path into this territory. This path is obviously not the only such and it likely is not the best, but we feel that it represents a creditable bit of exploration. In particular we suspect that the failure of the union and gap theorems will be features of any reasonable complexity theory for type-2.

There are obviously many open questions: What happens with the speedup theorems? What happens if we restrict all the β's to SB? What if we change to the unit cost model for OTMs? If we are stuck with naming large classes (e.g., the type-2 basic feasible functions) through unions, what are the general properties of these union classes. (Li [Li01] addresses many of these questions.) Going a little farther, one can ask: How can one extend our work to cover the effectively continuous type-2 functionals? (This requires a careful treatment of computation over partial (e.g., $\mathrm{N} \rightharpoonup \mathrm{N}$) arguments.) How can we extend this work beyond type-2? (Our notion of complexity bound seems amenable to realizer-based definitions of higher-type classes.)

References

[Blu67] M. Blum, *A machine-independent theory of the complexity of recursive functions*, Journal of the Association for Computing Machinery **14** (1967), 322–336.

[Bor72] A. Borodin, *Computational complexity and the existence of complexity gaps*, Journal of the Association for Computing Machinery **19** (1972), 158–174.

[CK90] S. Cook and B. Kapron, *Characterizations of the basic feasible functions of finite type*, Feasible Mathematics: A Mathematical Sciences Institute Workshop, (S. Buss and P. Scott, eds.), Birkhäuser, 1990, pp. 71–95.

[Con72] R. Constable, *The operator gap*, Journal of the Association for Computing Machinery **19** (1972), 175–183.

[Con73] R. Constable, *Type two computational complexity*, Proc. of the Fifth Ann. ACM Symp. on Theory of Computing, 1973, pp. 108–121.

[CU93] S. Cook and A. Urquhart, *Functional interpretations of feasibly constructive arithmetic*, Annals of Pure and Applied Logic **63** (1993), 103–200.

[HS65] J. Hartmanis and R. Stearns, *On the computational complexity of algorithms*, Transactions of the American Mathematical Society **117** (1965), 285–306.

[IKR01] R. Irwin, B. Kapron, and J. Royer, *On characterizations of the basic feasible functional, Part I*, Journal of Functional Programming (2001), to appear.

[Kap91] B. Kapron, *Feasible computation in higher types*, Ph.D. thesis, Department of Computer Science, University of Toronto, 1991.

[KC96] B. Kapron and S. Cook, *A new characterization of type 2 feasibility*, SIAM Journal on Computing **25** (1996), 117–132.

[Li01] C.-C. Li, *Type-2 complexity theory*, Ph.D. thesis, Syracuse University, 2001.

[MM69] E. McCreight and A. Meyer, *Classes of computable functions defined by bounds on computation*, Proc. of the First Ann. ACM Symp. on Theory of Computing, 1969, pp. 79–88.

[Odi89] P. Odifreddi, *Classical recursion theory*, North-Holland, 1989.

[Odi99] P. Odifreddi, *Classical recursion theory, volume II*, North-Holland, 1999.

[Rab60] M. Rabin, *Degree of difficulty of computing a function and a partial ordering of the recursive sets*, Report 2, University of Jerusalem, 1960.

[Rog67] H. Rogers, *Theory of recursive functions and effective computability*, McGraw-Hill, 1967, reprinted, MIT Press, 1987.

[Set92] A. Seth, *There is no recursive axiomatization for feasible functionals of type 2*, Seventh Annual IEEE Symposium on Logic in Computer Science, 1992, pp. 286–295.

[Set94] A. Seth, *Complexity theory of higher type functionals*, Ph.D. thesis, University of Bombay, 1994.

[You73] P. Young, *Easy constructions in complexity theory: Gap and speed-up theorems*, Proceedings of the American Mathematical Society **37** (1973), 555–563.