Feasibly Continuous Type-Two Functionals

Bruce M. Kapron
Department of Computer Science
University of Victoria
Victoria, BC
CANADA V8W 3P6
bmkapron@csr.uvic.ca

April 24, 1997

Abstract

A well-known theorem of type-two recursion theory states that a functional is continuous if and only if it is computable relative to some oracle. We show that a feasible analogue of this theorem holds, using techniques originally developed in the study of Boolean decision tree complexity.

1 Introduction

Type-two computability theory deals with the computability of functionals, which take functions and numbers as input, and produce numbers as output. A surprising and pleasing aspect of type-two computability is its close connections with topology on Baire space. Notions of *relative* type-two computability (that is, computability with respect to some oracle,) can be characterized using purely topological notions. In particular, a type-two functional is computable relative to an oracle if and only if it is continuous.

While the theory of type-two computability has been widely successful, relatively little work has been done on the development of a complexity theory for type-two functionals. The foundations of such a theory were laid by Constable (1973) and Mehlhorn (1976). Mehlhorn defined a class of *poly-time operators*, which were intended to be an analogue of the ordinary poly-time functions. The theory was advanced by Townsend (1990), who was the first to consider topological aspects of type-two complexity. Recently, Kapron and Cook (1996) gave a purely computational characterization of the class introduced by Mehlhorn, and Royer (1995) has considered a poly-time analogue of the effective operations.

In this paper, we extend the results of Townsend, and show that relative feasibility (in the sense of Mehlhorn) is equivalent to a kind of bounded continuity which we call *feasible* continuity. Our proof extends techniques originally developed in the study of relativized complexity classes and Boolean decision tree complexity.

2 Type-two computable functionals

A *(type-two) functional of rank* (k, l) is a mapping $F : (\mathbb{N} \to \mathbb{N})^k \to \mathbb{N}^l \to \mathbb{N}$. There are a number of notions of computability for type 2 functionals (see Rogers 1987). Here we will concentrate on the *computable functionals*, which are defined using the familiar oracle Turing machine (OTM) model. An oracle Turing machine of *rank* (k, l) is a multi-tape Turing machine with k *query input tapes* (read/write), k *query output tapes* (read-only), and some number of work tapes, one of which is designated as an input/output tape. An OTM M of rank (k, l) computes a functional F_M in the obvious way, with function inputs presented as oracles. To query the function input f_i at some input u, the binary representation of u is written on oracle input tape i, and oracle query state i is entered. This causes the value $f_i(u)$ to be written (in binary) on oracle output tape i. This operation is atomic (i.e. the machine cannot cause only part of $f_i(u)$ to be written.) A functional F is *computable* if there is an OTM M such that for all inputs $\vec{f}$ and $\vec{x}$, $F(\vec{f}, \vec{x}) = F_M(\vec{f}, \vec{x})$. In the sequel, we consider primarily functionals of rank $(1, 1)$. This simplifies the presentation, but none of our results depend on this restriction in an essential way.

It has long been known that there are close relationships between type-two computability and topology on Baire space. We will not discuss the details of this relationship here, but rather focus on one important correspondence. A functional F is *continuous* if for every f and x, there is some finite set Q

of query points such that, for all g, if $f(u) = g(u)$ whenever $u \in Q$, then $F(f, x) = F(g, x)$. A functional F is *computable in* a function f_0 if there is a computable functional G such that for all inputs f and x, $F(f, x) = G(f, f_0, x)$. The following result connects these two notions.

Theorem 2.1 *A functional F is continuous iff there is some function f such that F is computable in f.*

We have not been able to attribute this result to an individual, although it was clearly influenced by a number researchers working in recursion theory in the 1950's, and appears in various forms in the recursion theory literature of the time.

3 Feasible type-two computability and continuity

The first attempts to formulate a notion of polynomial-time computability for type-two functionals were made by Constable (1973) and Mehlhorn (1976). When considering polynomial-time type-two computability, we can use the OTM model described above, but it is necessary to formulate a convention on the number of time steps required for an oracle call. Since such a call is an atomic operation, it is plausible to assign it a unit cost. This, in fact, is the approach taken by Mehlhorn. We will use another plausible cost measure, namely: the cost of querying an oracle f at an input u is $|f(u)|$ where, for $x \in \mathbb{N}$, $|x|$ denotes the number of bits in the binary representation of x. So the cost of an oracle call is just the cost of receiving an answer from the oracle. In the sequel, we will consider only this *log-cost* model. Let $T_M(\vec{f}, \vec{x})$ denote the running time of M on inputs $\vec{f}$ and $\vec{x}$ in the log-cost model.

In Mehlhorn (1976), a function algebra intended to capture the notion of type-two poly-time computation is proposed. We will present a variant of this system, based on one given by Townsend (1990)[1].

Definition 3.1 A rank (k, l) functional F is obtained by

[1]The system of Townsend (1990) was formulated for computation over strings. We have made the straightforward modification to handle natural numbers. Also, we include all type-1 poly-time functions as initial functions. This simplifies the closure schemes required.

- *Composition* from $H, G_1, \ldots, G_l$ if

$$F(\vec{f}, \vec{x}) = H(\vec{f}, G_1(\vec{f}, \vec{x}), \ldots, G_l(\vec{f}, \vec{x}), \vec{x})$$

- *Expansion* from G if

$$F(\vec{f}, \vec{g}, \vec{x}, \vec{y}) = G(\vec{f}, \vec{x})$$

- *Limited recursion on notation* from G, H_0, H_1 and K if

$$\begin{array}{rcl} F(\vec{f}, \vec{x}, 0) & = & G(\vec{f}, \vec{x}) \\ F(\vec{f}, \vec{x}, 2y) & = & H_0(\vec{f}, \vec{x}, y, F(\vec{f}, \vec{x}, y)), \quad y > 0 \\ F(\vec{f}, \vec{x}, 2y+1) & = & H_1(\vec{f}, \vec{x}, y, F(\vec{f}, \vec{x}, y)) \\ |F(\vec{f}, \vec{x}, y)| & \leq & |K(\vec{f}, \vec{x}, y)| \end{array}$$

Let $F_1, \ldots, F_k$ be a set of functionals. The class $\mathrm{BFF}(F_1, \ldots, F_k)$ of functionals *basic feasible in* $F_1, \ldots, F_k$ is the smallest class containing all poly-time (type-1) functions, $F_1, \ldots, F_k$, and the *application functional* Ap define by $\mathrm{Ap}(f, x) = f(x)$, which is closed under composition, expansion and limited recursion on notation. The class BFF of *basic feasible functionals* is just $\mathrm{BFF}(\emptyset)$. For the sake of brevity, we will often refer to basic feasible functionals as *feasible functionals*, and we will say that BFF's are *feasibly computable*, or just *feasible*.

An explanation of the terminology BFF may be found in Cook and Kapron (1990) or Cook (1991).

The following result appears in Mehlhorn (1976), where it is proved for the unit-cost model. A proof for the log-cost model appears in Kapron (1991).

Theorem 3.2 *Let F be a type-two functional. F is a BFF iff there is an OTM M such that $F = F_M$ and a BFF G such that for every $\vec{f}$ and $\vec{x}$, $T_M(\vec{f}, \vec{x}) \leq |G(\vec{f}, \vec{x})|$.*

This results provides a computational interpretation of the basic feasible functionals.

Just as type-two computability gives rise to a notion of continuity, so does time-bounded type-two computability. In this setting, continuity is not only a matter of determination by a finite amount of information about function inputs, but rather of determination by a *bounded* amount of information. In general, this can be defined as follows.

Definition 3.3 Let F and B be rank $(1,1)$ functionals. F is *B-continuous* if for every f and x, there is some finite set $Q = \{u_1, \ldots, u_k\}$ of query points such that,

1. $k \leq B(f,x)$
2. $|u_i| \leq B(f,x), \quad 1 \leq i \leq k$
3. For all g, if $f(u) = g(u)$ whenever, $u \in Q$, then $F(f,x) = F(g,x)$.

If F is B-continuous for some B, we will say that F is *bounded continuous.* We will call B a *modulus functional.* A type-1 function f is *poly-bounded* if there is some polynomial p such that for all x, $|f(x)| \leq p(|x|)$. If B is of the form $\lambda f \lambda x.|G(f,g,x)|$ for some BFF G and some poly-bounded g, then we will say that F is *feasibly continuous.*

Note that the modulus functional B is used for two different kinds of bounding: both the number of query points and the size of individual query points must be bounded. The idea behind this is that if we have limited resources, we will not only be limited in the number of queries that we make, but we will also not be able to construct large query points. We may also want to bound the size of answers accepted from the oracle (this approach is taken in Townsend 1990). However, given the fact that we consider a query to be an atomic operation, we will not take the size of answers into consideration. The results presented here do not depend on this limitation.

Our definition of feasible continuity is an attempt to generalize the notion of bounding by a polynomial amount of information to type-two. Feasibility relative to a poly-bounded oracle is one reasonable way of formalizing such a generalization. Another possible generalization is considered in section 7.

Another way of viewing a bounded continuous functional F is in terms of the size of *certificates* which determine the value of F on given inputs f and x. A certificate is a set of query/answer pairs which is sufficient for determining the value of F. This may be formalized as follows:

Definition 3.4 A *finite function* is a partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ with $\mathrm{dom}(f)$ finite. The *size* of a finite function, denoted $||f||$, is the value

$$\max\{|\mathrm{dom}(f)|, \max_{x \in \mathrm{dom}(f)} |x|\}$$

Finite functions f and g are *compatible* if for every $x \in \mathrm{dom}(f) \cap \mathrm{dom}(g)$, $f(x) = g(x)$.

Suppose F is a rank $(1,1)$ functional, c is a finite function, and $x \in \mathbb{N}$. c is a *y-certificate for* F at x if for any $f : \mathbb{N} \to \mathbb{N}$, if $f(z) = c(z)$ for every $z \in \mathrm{dom}(c)$, then $F(f,x) = y$. We will say that c is a *certificate of* F if there are y and x such that c is a y-certificate for F at x.

The relationship between bounded continuity and certificate size is now clear:

Proposition 3.5 *A rank $(1,1)$ functional F is B-continuous if and only if for every f and x there is a $F(f,x)$-certificate $c \subseteq f$ for F at x with $||c|| \leq B(f,x)$*

What is the relationship between feasible computability and feasible continuity? A rank $(1,1)$ functional F is *feasibly bounded* if there is a BFF G and a poly-bounded g so that for all inputs f and x, $|F(f,x)| \leq |G(f,g,x)|$. It is not hard to show that if F is basic feasible in some poly-bounded f, then F is feasibly continuous and feasibly bounded. What about the converse of this implication? The standard proofs of 2.1 use unbounded search in order to find an appropriate certificate. This approach will not work for time-bounded computability. All is not lost, however. A converse of this implication may be obtained by extending results techniques which deal with 0-1 valued oracles, developed independently by a number of researchers (Blum and Impagliazzo 1987, Hartmanis and Hemachandra 1990, Tardos 1989), to the case of $\mathbb{N}$-valued oracles. We will now consider this approach.

4 Decision tree algorithms

Continuity is essentially a *non-deterministic* notion. That is, the value of a functional on some input depends on the *existence* of a certificate. However, there is no requirement that the certificate be obtained in a uniform way for every function input. On the other hand, computability relative to a fixed oracle is clearly a *deterministic* notion, since points at which the oracle is queried can depend only on the numerical input and oracle answers that have been already returned. This notion of deterministic computation may be formalized using *decision trees*. We present a decision tree model which is based on the standard model for Boolean functions.

Definition 4.1 A *decision tree algorithm* for computing a functional $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ takes a function f as input, queries f at a sequence $x_1, \ldots, x_k$ of *query points*, and returns an *answer* y such that

1. x_i depends only on the values $f(x_1), \ldots, f(x_{i-1})$ (in particular, the choice of x_1 is independent of the function input f.)

2. y depends only on the values $f(x_1), \ldots, f(x_k)$.

3. $y = F(f)$

If T is a decision tree algorithm computing a functional F, we will write $T(f)$ to denote the value $F(f)$. The *query cost* of a decision tree algorithm with respect to an input f is the number of points at which the algorithm queries f.

The following simple fact about certificates will prove essential in obtaining decision tree algorithms for functions with bounded certificates:

Lemma 4.2 *Suppose that c_y is a y-certificate for F at x and c_z is a z-certificate for F at x and $y \neq z$. Then there is some $x \in \mathrm{dom}(c_y) \cap \mathrm{dom}(c_z)$ so that $c_y(x) \neq c_z(x)$.*

Proof. If this were not the case, consider any $h : \mathbb{N} \to \mathbb{N}$ so that $c_y \cup c_z \subseteq h$. We have $F(h, x) = y$ and $F(h, x) = z$, a contradiction. QED

Theorem 4.3 *Let F be a rank $(1, 1)$ functional. For any x, s and v, there is a decision tree algorithm $T_{x,s,v}$ with the following properties, for any f:*

1. *If there is a $F(f, x)$-certificate $c \subseteq f$ for F at x of size at most s, then $T_{x,s,v}(f) = \mathrm{lmin}(F(f, x), v)$.*

2. *The query cost of $T_{x,s,v}$ with respect to f is s^2.*

Proof. Our proof is modeled on a proof for Boolean decision trees, given by Impaggliazzo and Naor (1988). Fix $x \in \mathbb{N}$. For every $s \in \mathbb{N}$, fix an enumeration of the certificates of F at x of size s, such that all the l-certificates are enumerated before any $(l + 1)$-certificate. Let lmin be the function such that $\mathrm{lmin}(x, y)$ is x if $|x| \leq |y|$, and y otherwise. The decision tree algorithm $T_{x,s,v}$ is given in figure 1.

- Let f_0 be the empty function and let i be 0.
- While $i \leq s$ and there is a certificate of size $\leq s$ compatible with f_i
  - If there is a y-certificate $c \subseteq f_i$, return $\mathrm{lmin}(y, v)$.
  - If all certificates of size $\leq s$ are y-certificates, return $\mathrm{lmin}(y, v)$.
  - Let c_i be the next certificate in the enumeration which is compatible with f_i.
  - For each $x \in \mathrm{dom}(c_i) - \mathrm{dom}(f_i)$, query f at x. Let f_{i+1} be f_i with the resulting query/answer pairs added.
  - Let i be $i + 1$.
- Return 0.

Figure 1: Decision tree algorithm $T_{x,s,v}$

We will first demonstrate the property (1) holds. Let $c \subseteq f$ be a $F(f,x)$-certificate for F at x of size at most s.

The proof consists of two parts. We first show that if the algorithm terminates with $i \leq s$, then $T_{x,s,v} = \mathrm{lmin}(F(f,x), v)$. We then show that the algorithm never terminates with $i > s$.

Suppose that the algorithm terminates with $i \leq s$. There are three ways in which this can happen. In the first case, we have some f_i for which there is no certificate of size at most s compatible with f_i. But clearly this is impossible because c is such a certificate. In the second case, there is some y-certificate $c' \subseteq f_i$. But then $T_{x,s,v}(f) = \mathrm{lmin}(y, v) = \mathrm{lmin}(F(f,x), v)$. In the third case, all certificates compatible with f_i of size at most s are y-certificates. Since $c \subseteq f$, c must be compatible with f_i and hence must be a y-certificate. But then $T_{x,s,v}(f) = \mathrm{lmin}(y, v) = \mathrm{lmin}(F(f,x), v)$.

Now suppose that the algorithm terminates with $i \geq s+1$. Suppose that c is a y-certificate and that c_s is a z-certificate. It must be the case that $z \leq y$, since for any finite $g \subseteq f$, c will be compatible with g. There are now two cases.

Case 1: $z < y$. By lemma 4.2, c_j and c must disagree on some input x_j, $0 \leq j < s$. *Claim:* We may assume that all the x_j are pairwise distinct. *Proof:* Suppose by way of contradiction that there are j and k with $j < k$ and $x_j = x_k$. Then $x_k \in \mathrm{dom}(f_{j+1})$. Since c_k is compatible with f_{j+1}, we have $c_k(x_k) = f_{j+1}(x_k) = f(x_k)$. Since $c \subset f$ and $x_k \in \mathrm{dom}(c)$, we also have $c(x_k) = f(x)$. Hence, $c_k(x_k) = c(x_k)$, which contradicts our choice of x_k, and so the claim follows. By hypothesis, $s \geq ||c|| \geq |\mathrm{dom}(c)|$. Along with the claim, this implies that $\mathrm{dom}(c) = \{x_0, \ldots, x_{s-1}\}$. Now, the procedure ensures that for $0 \leq i < s$, $x_i \in \mathrm{dom}(f_s)$. But for every $x \in \mathrm{dom}(c)$, $c(x) = f(x) = f_s(x)$, so that $c \subseteq f_s$. Recall that c is of size $\leq s$, so the algorithm terminates in iteration s by the success of the first test in the body of the while-loop This is a contradiction.

Case 2: $z = y$. Suppose by way of contradiction that for some $w > y$, there is an w-certificate c' compatible with f_s such that $||c'|| \leq s$. By lemma 4.2, for each j $0le$J $\leq s$, c_j and c' must disagree on some input x_j. *Claim:* We may assume that the x_j are pairwise distinct. The proof is as in the previous case. Therefore, by the claim $|\mathrm{dom}(c')| \geq s + 1$, which contradicts the hypothesis that $||c'|| \leq s$. Hence, no such c' exists. Therefore, in iteration s, all the certificates of size $\leq s$ compatible with f_i are y-certificates, so that the algorithm terminates by the the success of the second test in the body of the while-loop. This is a contradiction.

Property (2) can easily be established: since we terminate with $i \leq s$, at most s certificates are considered in the course of the algorithm. Since the size of certificates is bounded by s, at most s queries are made for each certificate considered. QED

5 Poly-bounded continuity and feasible computation

The result of the previous section may now be used in order to establish the desired equivalence between feasible continuity and feasibility in a poly-bounded oracle, for feasibly bounded functionals.

Theorem 5.1 *A rank $(1,1)$ functional F is feasibly continuous and feasibly bounded if and only if F is feasible in some poly-bounded oracle.*

Proof. Suppose that F is feasible in some poly-bounded g. In other words, there is a feasible functional G and a poly-bounded function g such that for all f and x, $F(f,x) = G(f,g,x)$. Suppose that G is computed by an OTM M with running time bounded by $|H(f,g,x)|$, on all inputs f,g and x, for some BFF H. With such a time bound, M can make at most $|H(f,g,x)|$ queries to f, and each query input must have size bounded by $|H(f,g,x)|$. Suppose on inputs f and x, M with oracle g queries f at $\sigma = x_1,\ldots,x_k$. Then for and f' which agrees with f on σ, $F(f',x) = F(f,x)$. By the preceding argument, $k \leq |H(f,g,x)|$ and for each x_i, $|x_i| \leq |H(f,g,x)|$, so F is feasibly continuous. It also follows from the bound on the running time of M that $|F(f,x)| \leq |H(f,g,x)|$, for all inputs f and x, so that F is feasibly bounded.

Now suppose F is feasibly continuous, as witnessed by the BFF H_0 and the poly-bounded function h_0 and feasibly bounded, as witnessed by the BFF H_1 and the poly-bounded function h_1. We will show that there is a BFF G and a poly-bounded function g so that for all inputs f and x,

$$F(f,x) = G(f,g,x,H_0(f,h_0,x),H_1(f,h_1,x)). \tag{1}$$

Using the existence of poly-time pairing and projection functions, and the fact that the BFF's are closed under functional substitution (see, e.g., Townsend 1990), it follows that F is basic feasible in some poly-bounded oracle.

We begin by defining the poly-bounded oracle g. We assume the existence of standard, poly-time, pairing and sequence encoding functions. The code of the empty sequence is denoted $()$ and the code of a sequence $x_1,\ldots,x_k$ is denoted $(x_1,\ldots,x_k)$. For each x, g is defined using the algorithm $T_{x,s,v}$, as follows:

$$\begin{aligned}
g(\langle x, 2^s-1, v, ()\rangle) &= 2x_1+1; \\
g(\langle x, 2^s-1, v, (f(x_1),\ldots,f(x_i))\rangle) &= 2x_{i+1}+1, \quad i<k; \\
g(\langle x, 2^s-1, v, (f(x_1),\ldots,f(x_k))\rangle) &= 2y,
\end{aligned}$$

where $x_1,\ldots,x_k$ is the sequence of queries generated by $T_{s,x,v}$ on input f, and $y = T_{x,s,v}(f)$. For any other input u, $g(u) = 0$. By the construction of $T_{x,s,v}$, $|x_i| \leq s$ and $|y| \leq |v|$, so that g is poly-bounded.

We now define a BFF G which satisfies condition (1). Define the BFF H as follows:

$$H(u,s,v) = \begin{cases} u & \text{if } |u| \leq \max\{2|s|+1, 2|v|\} \\ 0 & \text{otherwise.} \end{cases}$$

Now consider the following definition of a functional G', using a course-of-values primitive recursion.

$$
\begin{aligned}
G'(f,h,x,s,v,0) &= H(h(\langle x, 2^{|s|}-1, v, \sigma_0\rangle), s, v) \\
G'(f,h,x,s,v,t+1) &= \begin{cases} H(h(\langle x, 2^{|s|}-1, v, \sigma_{t+1}\rangle), s, v) & \text{if } u_t \text{ is odd} \\ u_t & \text{otherwise,} \end{cases}
\end{aligned}
$$

where $u_i = G(f,h,x,s,v,i)$, $\sigma_0 = ()$, and $\sigma_{i+1} = (f((u_0-1)/2), \ldots, f((u_i-1)/2))$.

Using property (2) of $T_{x,s,v}$, it is not hard to show that if we let

$$G(f,h,x,s,v) = G'(f,h,x,s,v,|s|^2)/2,$$

then G satisfies the correctness condition (1). Finally, we claim that G is a BFF. We recall that there is a poly-time function $\#$ with the property that $|x \# y| = |x| \cdot |y| + 1$, so that G may be defined using a course-of-values primitive recursion on the *length* of its recursion parameter. Moreover, since

$$|G(f,h,x,s,v)| \le \max\{2|s|+1, 2|v|\},$$

we may translate this definition of G into a definition using LRN. QED

6 Related work

In Townsend (1990), feasible continuity is compared to *nondeterministic* relative feasible computability. This requires the notion of a *nondeterministic oracle Turing machine* (NOTM). Such a machine may have more than one possible computation path for given function and number inputs. A NOTM M *computes* a rank (k,l) functional F if for all inputs $\vec{f}$ and $\vec{x}$, every computation of M which reaches a halting state does so with the value $F(\vec{f},\vec{x})$ written on its answer tape. A NOTM M is *feasible* if there is a BFF G such that for all inputs $\vec{f}$ and $\vec{x}$, if M halts then the running time of M is bounded by $|G(\vec{f},\vec{x})|$. A functional F is *feasibly nondeterministically computable* if there is a feasible NOTM which computes F. For a function g, F is *feasibly nondeterministically computable in g* is there is a feasibly nondeterministically computable G such that for all $\vec{f}$ and $\vec{x}$, $F(\vec{f},\vec{x}) = G(\vec{f},g,\vec{x})$.

Proposition 4.8 of Townsend (1990) can now be stated (using our terminology):

Proposition 6.1 (Townsend) *If a functional F is feasibly bounded and has a modulus functional basic feasible in some g then it is feasibly nondeterministically computable in some g.*

It is conjectured that the converse of this result also holds. It is not clear how this is related to our theorem 5.1, especially since this result uses arbitrary oracles, as opposed to poly-bounded ones.

Theorem 2.1 has an *effective* version. In order to state it, we need to define an effective version of continuity.

Definition 6.2 A rank $(1,1)$ functional F is *effectively continuous* if there is a computable function $g : \mathbb{N}^2 \to \mathbb{N}$ so that for all inputs f and x, $F(f,x) = k$ if there is some u, which is the code of some finite $f_0 \subseteq f$ under a fixed effective encoding scheme, such that $g(u,x) = k$.

The effective version of theorem 2.1 can now be stated.

Theorem 6.3 *A rank $(1,1)$ functional F is effectively continuous iff it is computable.*

It is not hard to also formulate an effective version of theorem 5.1, using a suitable notion of effective feasible continuity. However, Royer (1996) has shown that if every effectively feasibly continuous functional is feasible, then a certain complexity-theoretic hypothesis regarding the search power of the class P fails. However, he also shows that this hypothesis holds relative to some oracle. Thus, proving an effective version of theorem 5.1 will be difficult using existing techniques.

7 Conclusions and future work

We have shown that a feasible version of a fundamental theorem relating relative computability and continuity holds. The techniques used to prove the feasible version differ considerably from those used to prove the original result, and rely on a much finer computational analysis of the related notions. The approach used is an extension of a known result in Boolean decision tree complexity.

With appropriate formulations of other higher-type complexity classes, we could apply 4.3 to obtain versions of our main theorem 5.1 for functionals with a modulus from that class. One important property of BFF's which we use in the proof of 5.1 is that since F is feasibly continuous and feasibly bounded, we can feasibly compute the values s and v which are needed in order to apply 4.3.

In Kapron and Cook (1996), a characterization of feasible functionals was given using time-bounded OTM computability. This characterization uses bounding functionals which, in general, are not feasibly computable. We now give a brief summary of this characterization.

Definition 7.1 If $f : \mathbb{N} \to \mathbb{N}$ is a type-1 function, the *length of* f is the functions $|f| : \mathbb{N} \to \mathbb{N}$ defined by $|f|(n) = \max_{|x| \leq n} |f(x)|$.

The function $|f|$ may be viewed as giving the uniform worst-case complexity of accessing f in the log-cost model.

Definition 7.2 A *second-order polynomial* over type-2 variables $f_1, \ldots, f_k$ and type-1 variables $x_1, \ldots, x_l$ is an expression $\mathbf{p}$ of one of the following forms:

- c (a numerical constant)
- x_i, $1 \leq i \leq l$
- $\mathbf{p}_1 + \mathbf{p}_2$
- $\mathbf{p}_1 \cdot \mathbf{p}_2$
- $f_i(\mathbf{p}_1)$, $1 \leq i \leq k$,

where $\mathbf{p}_1$ and $\mathbf{p}_2$ are second-order polynomials over the variables in question. For functions $g_i : \mathbb{N} \to \mathbb{N}$, $1 \leq i \leq k$ and numbers $n_1, \ldots, n_l$, $\mathbf{p}(\vec{g}, \vec{n})$ denotes a natural number, defined in the obvious way.

A functional F is *poly-time* if there is an OTM such that $F = F_M$ and for all inputs $\vec{f}$ and $\vec{x}$, $T_M(\vec{f}, \vec{x}) \leq \mathbf{p}(|f_1|, \ldots, |f_k|, |x_1|, \ldots, |x_l|)$. F is *poly-time in* a function f_0 if there is a poly-time functional G such that for all $\vec{f}$ and $\vec{x}$, $F(\vec{f}, \vec{x}) = G(f_0, \vec{f}, \vec{x})$.

The characterization of Kapron and Cook may now be stated:

Theorem 7.3 *Let F be a type-2 functional. F is a BFF iff F is poly-time.*

This characterization leads us to consider another class of continuous functionals.

Definition 7.4 A rank $(1,1)$ functional F is *poly-continuous* if it has a modulus functional of the form $\lambda f \lambda x.\mathbf{p}(|f|,|x|)$ for some second-order polynomial $\mathbf{p}$. F is *poly-bounded* if there is a second-order polynomial $\mathbf{p}$ such that $|F(f,x)| \leq \mathbf{p}(|f|,|x|)$ for all inputs f and x.

We conjecture that a functional is poly-continuous and poly-bounded if and only if it is feasible in some oracle. However, in order to prove this conjecture we will need to go beyond the techniques used in the proof 5.1, precisely because we cannot feasibly compute the values s and v using the modulus and bounding functionals, since these will not, in general, be feasibly computable (see Kapron and Cook 1996, theorem 5.5.) The correctness of $T_{x,s,v}$ depends heavily on the fact that we know that a certificate of size at most s exists. In particular, in the case where every compatible certificate is a y-certificate, we use the bound on certificate size to return a value $\mathrm{lmin}(y,v)$, without having actually constructed a certificate. This "nonconstructivity" becomes a much more serious issue for poly-continuity, precisely because we cannot precompute s.

Acknowledgements

I would like to thank my wife, Valerie King, for pointing out the relevance of techniques from Boolean decision tree complexity to this problem. I would also like to thank Russell Impagliazzo and Yiannis Moschovakis for providing historical information. Finally, I would like to thank Jim Royer for pointing out an error in the original version of the main theorem, for suggesting improvements to the proof of theorem 4.3, and for many helpful discussions.

References

M. Blum and R. Impagliazzo, Generic oracles and oracle classes, in *Proc. 28th Ann. IEEE Symp. Found. Comput. Sci.*, 1987, 118–126.

R. CONSTABLE, Type 2 computational complexity, in *Proc. 5th Ann. ACM Symp. Theor. Comput.*, 1973, 108–121.

S. A. COOK, Computability and complexity of higher type functions, in *Logic from Computer Science*, ed. Y. N. MOSCHOVAKIS, Springer-Verlag, 1991, 51–72.

S. A. COOK AND B. M. KAPRON, Characterizations of the basic feasible functionals of finite type, in *Feasible Mathematics*, ed. S. BUSS AND P. J. SCOTT, Birkhauser, 1990, 71–93.

J. HARTMANIS AND L. A. HEMACHANDRA, Robust machines accept easy sets, *Theoret Comput. Sci.*, **74** (1990), 217–225.

R. IMPAGLIAZZO AND M. NAOR, Decision trees and downward closures, in *Proc. 3rd Ann. IEEE Struct. Complexity Theor.*, 1988, 29–38.

B. M. KAPRON, *Feasible Computation in Higher Types*, PhD. Thesis, Department of Computer Science, University of Toronto, 1991.

B. M. KAPRON AND S.A. COOK, A new characterization of type-2 feasibility, *SIAM J. Comput.*, **25** (1996), 117–132.

K. MEHLHORN, Polynomial and abstract subrecursive classes, *J. Comput. System Sci.*, **12** (1976), 147–178.

H. ROGERS, *Theory of Recursive Functions and Effective Computability*, MIT Press, Cambridge, 1987.

J. S. ROYER, Semantics vs. syntax vs. computations: machine models for type-2 polynomial-time bounded functionals, *Proc. 10th Ann. IEEE Struct. Complexity Theor.*, 1995, 37–48.

J.S. ROYER, personal communication, 1996.

G. TARDOS, Query complexity, or why is it difficult to separate $NP^A \cap co-NP^A$ from P^A by a random oracle?, *Combinatorica*, **9**, (1989), 385–392.

M. TOWNSEND, Complexity for type-2 relations, *Notre Dame J. Formal Logic*, **31** (1990), 241–262.