Resource-bounded Continuity and Sequentiality for Type-two Functionals

SAMUEL R. BUSS
University of California, San Diego

and
BRUCE M. KAPRON
University of Victoria

We define notions of resource-bounded continuity and sequentiality for type-two functionals with total inputs, and prove that in the resource-bounded model there are continuous functionals which cannot be efficiently simulated by sequential functionals. We also show that for some naturally-defined classes of continuous functionals, an efficient simulation is possible.

Categories and Subject Descriptors: F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes—*relations among complexity classes*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*relations between models*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*computability theory*

General Terms: Theory, Algorithms

Additional Key Words and Phrases: Higher-order complexity, sequential computation, decision trees

1. INTRODUCTION

The notion of *continuity* has long played an important role in higher-type computability theory, as well as in the theory of programming languages. A central theme in this area is the relationship between continuity and *sequentiality*. Continuity can be seen as a proper generalization of sequentiality, although there are certain special cases for which the two notions coincide, for example when considering type-two functionals with *total* function inputs. The primary purpose of this paper is to point out that, even in the special case of total functions, continuity is in fact more powerful then sequentiality when the quantitative consideration of

Samuel R. Buss, Department of Mathematics, University of California, San Diego, La Jolla, CA 92093-0112, sbuss@ucsd.edu. Partially supported by NSF grant DMS-9803515 and by a cooperative research grant INT-9600919/ME-103 from the NSF (USA) and the MŠMT (Czech Republic).

Bruce M. Kapron, Computer Science Department, University of Victoria, Victoria BC, CANADA V8W 3P6, bmkapron@csr.uvic.ca. Partially supported by NSERC Research Grant OGP0138744.

how much an input function must be queried is admitted.

Early models of continuous higher-type functionals were proposed by Kleene [1959a] and Kreisel [1959]. Kleene used the term *countable* rather than continuous, reflecting the fact that continuous functionals have countable representations. We discuss Kleene's approach in more detail in section 1.1 below. At the time of Kleene and Kreisel's work, sequential higher-type computability was primarily understood in terms of Kleene's schemas S1-S9 [Kleene 1959b],[Kleene 1963]. All of these models defined classes of functionals with total input. It was readily apparent that at type level two, the sequential and continuous functionals on total inputs coincided. By later results of Tait [1962] and of Gandy and Hyland [1977] it is the case that at type level three, there are *effectively* countable functionals which are not S1-S9 computable (over countable inputs).

More recently, the relative power of continuous and sequential functionals on total inputs has been considered by Berger [1993], Plotkin [1999] and Normann [1998]. Berger [1993] conjectured that all effective elements in the hierarchy of total continuous functionals were in fact definable in (sequential) PCF. Plotkin [1999] proved a special case of the conjecture, which is proved in full generality in Normann [1998]. Recently, new models of sequential higher-order functionals have been proposed by Longley [1998] and van Oosten [1999]. Van Oosten's model provides a simple combinatorial structure which makes obvious the connection between sequential functionals and familiar notions of decision trees from computational complexity theory.

While the results in this paper pertain to models for computation in higher types, the techniques used are closely related to work in Boolean decision tree complexity. Namely, a technique known as "Blum's trick" ([Blum and Imagliazzo 1987; Hartmanis and Hemachandra 1990; Tardos 1989]), which is used to show that any Boolean functions with small nondeterministic and co-nondeterministic complexity has a small-depth decision tree is generalized to show that in certain cases, sequential functionals can efficiently simulate continuous functionals. A lower bound on Boolean decision trees for *search* problems [Impagliazzo and Naor 1988] is the basis for our main result, showing that in general, not all type-two continuous functionals with total inputs can be efficiently simulated by sequential functionals. Note that the results presented here are about type-two functionals in the full type hierarchy. We do not consider models in which type-one inputs are computable.

1.1 The Countable Functionals

Kleene's characterized the countable functionals as a subclass of the hereditarily total functionals which in some sense depended on only a finite amount of information about their inputs. Formally, this was achieved through the use of *associates*, type-one functions used to encode higher-type objects.

Definition 1.1. Suppose that $F : (\mathbb{N} \to \mathbb{N}) \times \mathbb{N} \to \mathbb{N}$ is a type-two functional. $\alpha : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is an *associate* for F if for every f and x there is some $k > 0$ such that:

—For all $j < k$, $\alpha(\overline{f(j)}, x) = 0$

—For all $l \geq k$, $\alpha(\overline{f(l)}, x) = F(f, x) + 1$, where $\overline{f(i)}$ denotes the encoding $\langle f(0), \ldots, f(i-1)\rangle$. A functional is *countable* just in case it has an associate. If α is an associate, let F_α denote the countable functional that it defines.

Kleene's definition extends to higher types in the obvious way. Once we have defined associates for functionals at type level n, associates at type level $n+1$ are functions whose inputs are encodings of finite initial segments of associates for functionals at type level n.

Note while Kleene's approach is extensional, in that it identifies the countable functionals as a subclass of the collection of all hereditarily total functionals through the use of associates, Kreisel took an intensional approach which begins with functions defined on "formal neighbourhoods" and obtains the continuous functionals as an extensional collapse of these. Nevertheless, the restriction of Kreisel's continuous functionals to pure (non-mixed) types coincides with Kleene's countable functionals.

We refer the reader to the surveys [Cook 1990],[Gandy and Hyland 1977] and [Longley 2001] for more information on these issues and higher-type computability in general.

1.2 Motivation

Cook and Kapron [1990] and Kapron [1991] developed a theory of feasible higher-order computation over the full hierarchy of hereditarily total functionals. In doing so they observed the difficulty of extending the characterizations possible at type level two to higher types. A possible solution to these problems would be to work over the hierarchy of continuous higher-type functionals. In particular, since these functionals can be represented in a natural way using type-one functions (i.e., the *associates*), there would be some hope of reducing a theory of higher-order feasibility to type-two feasibility. However, a naive approach to this reduction does not work, as we will now show.

We could now propose to define the feasible type-two functionals as those which have a (type-one) poly-time associate. But consider the following α:

$$\alpha(s,n) = \begin{cases} s_{2^n-1} & \text{if } s = \langle s_0, \ldots s_{k-1}\rangle \text{ and } k \geq 2^n; \\ 0 & \text{otherwise} \end{cases}$$

Then α is poly-time. However $F_\alpha(\lambda x.x, n) = 2^n - 1$, which contradicts the basic assumption that any class of feasible functionals must be closed under functional substitution.

The preceding example demonstrates that we cannot define a notion of type-two feasibility based on continuous functionals without considering the modulus of continuity.[1] The definition given below in 3.4 is based on such a notion of modulus functional. Once this is done it is natural to consider the relationship between resource-bounded continuity and sequentiality. This leads to the definition of resource-bounded sequentiality given below in 3.6.

[1]A similar observation, with respect to feasible real analysis, is made by Ko [1986].

2. PRELIMINARIES

Definition 2.1. We denote by $\mathbb{N} \rightharpoonup \mathbb{N}$ the set of all partial functions from $\mathbb{N}$ to $\mathbb{N}$ and by $\mathbb{N} \to \mathbb{N}$ the set of all total functions. If $f : \mathbb{N} \rightharpoonup \mathbb{N}$, then $\mathrm{dom}(f)$, the *domain* of f is the set of all $z \in \mathbb{N}$ for which $f(z)$ is defined. A *finite function* is a partial function $c : \mathbb{N} \rightharpoonup \mathbb{N}$ with finite domain. A *type-two functional of rank* (k, l) is a mapping $F : (\mathbb{N} \rightharpoonup \mathbb{N})^k \times \mathbb{N}^l \rightharpoonup \mathbb{N}$. In the sequel, we will typically consider only total rank $(1, 1)$ functionals with total inputs (i.e., functionals $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$), and use the term *type-two functional* to describe such.

As described above, our motivation in considering continuous functionals is motivated by Kleene's definition of associate. However, we will use the following alternate definition of continuity. It is not hard to see that this is equivalent to an associate-based definition.

Definition 2.2. A type-two functional F is *continuous* if for every $f : \mathbb{N} \to \mathbb{N}$ and $x \in \mathbb{N}$, there is a finite set $Q \subseteq \mathrm{dom}(f)$ such that for all functions $g : \mathbb{N} \to \mathbb{N}$, if $f(z) = g(z)$ for all $z \in Q$, then $F(f, x) = F(g, x)$.

Van Ooosten [1999] introduces a class of sequential functionals over all finite types, based on a combinatory algebra of partial functions from $\mathbb{N}$ to $\mathbb{N}$. We will follow his approach in defining sequential type-two functionals This definition is based on the idea of a "dialogue" between two functions g and f. g queries f at a sequence of inputs, with each query being determined only by answers to previous queries. After a sufficient number of successful queries, g produces a final answer. Our definitions here are almost identical to those of [van Oosten 1999], although our final definition of sequentiality is for the more general case of functionals which may take functions as well as numbers as input and as noted above, we will restrict our attention to total functionals with total inputs.

Definition 2.3. Let $\langle\cdot\rangle : \mathbb{N}^* \to \mathbb{N}$ be an efficient sequence encoding function. An encoding $u = \langle u_0, \ldots, u_{n-1}\rangle$ of a sequence is a *dialogue* between $g : \mathbb{N} \rightharpoonup \mathbb{N}$, $f : \mathbb{N} \to \mathbb{N}$ if for all i, $0 \le i \le n-1$, there is a j such that $g(u^{<i}) = 2j$ and $f(j) = u_i$, where $u^{<i} = \langle u_0, \ldots, u_{i-1}\rangle$. The *application* $g|f$ is defined with value y (written $g|f = y$), if there is a dialogue u between g and f such that $g(u) = 2y+1$. If $g(u^{<i}) = 2j$ we will say that g *queries* f *at* j. If $g(u) = 2y+1$, we will say that g *answers* y. For a given g and f, we do not rule out the possibility that there is no dialogue between g and f (e.g., g may never answer, or may reach a point where it does not have a query). However, Note that there can be no more than one possible dialogue between f and g.

If $g : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$ and $x \in \mathbb{N}$, let g_x denote the function $\lambda z.g(x, z)$. Every such g determines a type-two functional F_g defined by

$$F_g(f, x) = y \text{ iff } g_x|f = y.$$

A type two functional $F : (\mathbb{N} \to \mathbb{N}) \times \mathbb{N} \to \mathbb{N}$ is *sequential* if $F = F_g$ for some $g : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$.

Suppose $F = F_g$ is a sequential functional. Then for any f and x, let

$$\mathrm{Queries}(g, f, x) = \{\tfrac{1}{2} g_x(u^{<i}) \mid 0 \le i < n-1\},$$

where $\langle u_1, \ldots, u_{n-1}\rangle$ is the dialogue between g_x and f.

Remark 2.4. The reader might wonder why we are giving first-class status to natural numbers, rather than working with pure types (representing $x \in \mathbb{N}$ as $\lambda y.x$). While the latter approach would be possible, in our opinion the benefit of working directly with numbers outweighs the drawback of a mixed type structure.

We note the following alternate characterization of sequentiality:

PROPOSITION 2.5. *A type-two functional F is sequential iff there is an oracle Turing machine M and a function g such that $F = M^g$.*

PROOF: Suppose $F = F_g$. It is easy to construct an oracle Turing machine M so that M^g on inputs f and x just follows the dialogue between g_x and f to compute $F(f, x)$. The converse follows directly from the sequential nature of Turing machine computation. □

It is well-known that if we do not restrict our attention to total functionals with total inputs, there are continuous type-two functionals which are not sequential, e.g.,

$$F(f, x) = \begin{cases} 1 & \text{if } f(x) = 1 \text{ or } f(x+1) = 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

On the other hand, if we only consider type-two functionals restricted to total inputs, it has long been known that continuous and sequential type-two functionals coincide, at least since the time of Kleene's work on the countable functionals [Kleene 1959a]. A proof of this result is given in the next section. Note that early formulations of this result did not refer explicitly to sequential functionals as presented here, but to the equivalent notion of computability in an oracle.

3. RESOURCE-BOUNDED CONTINUITY AND SEQUENTIALITY

We now turn to resource-bounded versions of the definitions from the previous section. We first give definitions of these new notions, and then provide some background to motivate the definitions.

3.1 Definitions

The efficiency of continuous and sequential functionals will be defined in terms of the amount of information they must obtain about function inputs in order to produce an answer. While this obviously must account for the number of input values at which the function input must be queried, it is also reasonable to also account for the actual size of input values as well. This is formalized as follows.

Definition 3.1. Suppose F is a functional, $x, y \in \mathbb{N}$, and c is a finite function. Then c is a *y-certificate* for F, x if $F(f, x) = y$ for every $f \supseteq c$. When the actual value y is not important, we will just say that c is a *certificate* for F, x. The *size* of a certificate c, denoted $|c|$, is $\Sigma_{z \in \mathrm{dom}(c)} |z|$ where for $x \in \mathbb{N}$, $|x|$ denotes the length of the dyadic notation of x (so, in particular $|0| = 0$).

Remark 3.2. The definition of the size of a certificate is intended to capture the cost of accessing a function input f in a black-box manner, i.e., "how much" of f must be examined in order to determine $F(f, x)$. The given definition accounts not only for the number of queries that must be made to f, but also the size of the queries. This reflects our intuition that it should be more costly to construct larger inputs at which to query f. Note that our definition does not count the cost of answers from f as part of the overall cost. This choice is somewhat arbitrary. However, in the sequel we will normally bound the size of certificate uniformly in f, in such a way that the cost of the size of answers from f may always be accounted for in the bound.

The definition of continuity may now be rephrased using certificates:

PROPOSITION 3.3. *A functional F is continuous iff for all functions f and $x, y \in \mathbb{N}$, $F(f, x) = y$ iff there is a y-certificate c for F, x such that $c \subseteq f$.*

By using certificate size, it is possible to define a more refined version of continuity.

Definition 3.4. Suppose F, B are functionals. F is *B-continuous* if for all functions f and $x, y \in \mathbb{N}$, $F(f, x) = y$ iff there is a y-certificate g for F, x such that $g \subseteq f$, and $|g| \leq B(f, x)$. In this case, we will also say that B is a *modulus of continuity for F*. If $\mathcal{F}$ is a collection of functionals, then F is *$\mathcal{F}$-continuous* if it is B-continuous for some $B \in \mathcal{F}$.

We can also introduce a resource-bounded version of sequentiality, based on the notion of the length of a dialogue.

Definition 3.5. The *length* of a dialogue $u = \langle u_0, \ldots, u_{n-1} \rangle$, between g and f, denoted $|u|$, is $\sum_{i=0}^{n-1} \left| \frac{1}{2} g(u^{<i}) \right|$.

Definition 3.6. Suppose F, B are functionals. F is *B-sequential* if there is a function g such that for all functions f, $F(f, x) = y$ iff $g_x|f = y$ via a dialogue u between g_x and f, such that $|u| \leq B(f, x)$. If $\mathcal{F}$ is a collection of functionals, then F is *$\mathcal{F}$-sequential* if it is B-sequential for some $B \in \mathcal{F}$.

Remark 3.7. The motivation behind our definition of the length of a dialogue is essentially the same as that for the definition of the size of a certificate.

We now prove a general result relating resource-bounded continuity and resource-bounded sequentiality.

THEOREM 3.8. *Any B-continuous functional is B'-sequential, where B' is defined by:*

$$B'(f, x) = B(f, x) \cdot 2^{B(f,x)}.$$

PROOF: Suppose F is B-continuous. Given f and x, for $0 \leq i \leq 2^{B(f,x)} - 1$, let c_i be defined as follows:

$$c_i(u) = \begin{cases} f(u) & \text{if } u < i; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Since F is B-continuous, it must be the case that some c_i is a certificate for F, x. Now we can design a g such that for inputs f and x, g_x successively queries f at $0, 1, \ldots$ until it has found $c_i \subseteq f$ which is a y-certificate for F, x, and then returns y. Note that this results in a dialogue consisting of at most $2^{B(f,x)}$ queries, with each query having size at most $B(f, x)$. □

The method for constructing g in this proof is essentially Kleene's method of associates. While it is perhaps just a matter of stating the obvious, to our knowledge no mention of the inefficiency of this construction is made in the computability theory literature. However, once we have noted this inefficiency, it is natural to ask whether a more efficient construction is possible.

In the main result of this paper, we show that in general, an efficient construction is not possible, by demonstrating a B-continuous functional, which is not $\lambda f \lambda x.(B(f,x))^n$-sequential for any $n \geq 0$. Before proving this result, however, we show that in the case where the continuous functional in question has a modulus functional B which is itself C-sequential, a construction which is efficient, relative to B *and* C, is possible. This implies that, for a certain natural class of moduli, continuity and sequentiality do coincide.

4. AN EFFICIENT SIMULATION

The result presented in this section is a generalization of a technique known as "Blum's trick", ([Blum and Imagliazzo 1987; Hartmanis and Hemachandra 1990; Tardos 1989]), which relates certificate size and decision tree complexity for Boolean functions, to the case of the type-two functionals considered here. A similar proof is given in [Kapron 1999], but for a special case which is described in detail below.

We begin with the following simple fact about certificates, which will prove essential in obtaining decision tree algorithms for functions with bounded certificates:

LEMMA 4.1. *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. □

THEOREM 4.2. *Suppose that F is B-continuous, and B is C-sequential. Then F is $\lambda f \lambda x.(C(f,x) + [B(f,x)]^2)$-sequential.*

PROOF (sketch): We must show that there is a function g with the following properties.

—For every f and x, $g_x | f = F(f,x)$.

—For every f and x, the dialogue u between g_x and f has length bounded by $C(f,x) + [B(f,x)]^2$.

We begin by using Blum's trick to construct functions $g_{x,s}$ such that for any f and x, if there is a y-certificate $c \subseteq f$ for F, x with size bounded by s, then

(*). $g_{x,s} | f = y$

(**). the dialogue u between $g_{x,s}$ and f has length bounded by s^2.

Once we are able to construct $g_{x,s}$, we can then construct g as follows: g, when given inputs f and x, first computes $B(f,x)$ via a dialogue of length $C(f,x)$. This is possible since B is C-sequential. Having done so, it then proceeds as $g_{x,B(f,x)}$.

It remains to give the construction of $g_{x,s}$. More specifically, we will show to construct a dialogue between $g_{x,s}$ and an arbitrary f. We can assume that $g_{x,s}$ is undefined for any input which does not arise from some such dialogue. We begin by fixing an enumeration of all certificates for F, x of size at most s, such that all z-certificates appear before any $(z+1)$-certificates. We will construct a sequence $g^1, g^2, \ldots$ of functions such that $g_{x,s} = \bigcup_i g^i$. At the same time, we will define $f^1, f^2, \ldots$ where $\mathrm{dom}(f^i)$ consists of all queries made by g^i, and $u^1, u^2, \ldots$ where u^i is the (partial) dialogue between g^i and f.

Remark 4.3. The definition of g^i is completely determined by the values of f at which it has been queried so far. Thus $g_{x,s}$ is well-defined, since the construction would proceed identically for any f' for which $\mathrm{Queries}(g_{x,s}, f', x)$ and $\mathrm{Queries}(g_{x,s}, f, x)$ are equal.

To start, we have g^0 and f^0 everywhere undefined, and u^0 the empty sequence.

At stage $i > 0$, suppose that g^{i-1}, f^{i-1} and u^{i-1} have been defined. We define g^i to be a minimal extension of g^{i-1}, according to which one of the following four cases hold

(1) If there are no remaining certificates which are consistent with f^{i-1}, then $g^i(u^{i-1})$ is undefined; the construction terminates at this point (so that in fact $g^i = g^{i-1}$).

(2) If there is any z-certificate $c \subseteq f^{i-1}$, then $g^i(u^{i-1}) = 2z+1$ and $t = i$; the construction terminates at this point.

(3) If all certificates consistent with f^{i-1} are z-certificates, then $g(u^{i-1}) = 2z+1$ and $t = i$; the construction terminates at this point.

(4) Otherwise, let c^i be the next certificate in the enumeration consistent with f^{i-1}, and suppose that $\mathrm{dom}(c^i) = \{x_1, \ldots, x_q\}$, $q \leq s$. Then $g^i(u^{i-1}) = 2x_1$ and for $1 \leq j < q$,

$$g^i(u^{k-1\frown}\langle f(x_1), \ldots, f(x_j)\rangle) = 2x_{j+1},$$

where $\frown$ denotes concatenation, $u^i = u^{i-1\frown}\langle f(x_1), \ldots, f(x_q)\rangle$, Also let $f^i = f^{i-1} \cup \{\langle x, f(x)\rangle \mid x \in \mathrm{dom}(c^i)\}$

Before showing that $g_{x,s}$ satisfies the two conditions given above, we give an intuitive explanation and justification for the construction. We would like to find a certificate $c \subseteq f$, which will determine the value of $F(f,x)$. Supposing that we know that there is such a certificate of size s, how do we find it? Basically, we will use the fixed enumeration of certificates to pin down more and more of f, until we find c or at least find the value for $F(f,x)$ which is given by c. This is what happens in case 4: we find the next consistent certificate, and query f at a point in the domain of this certificate. So u^i is defined to be the extension of u^{i-1} by the answers to the queries determined by the chosen certificate, and f^{i-1} is extended to f^i to account for the queries. Querying f in this way serves a two-fold purpose. First of all, we may now have uncovered enough of f to be able to see that there is a certificate $c \subseteq f$ in which case we are done, or that the only remaining certificates that are consistent with f are z-certificates, in which case we are also done, since we are assuming that there is at least one certificate $c \subseteq f$ of size s. The second purpose served by querying f is to "knock out" certificates which do not agree with what has so far been seen of f. The heart of the argument below is that if there is indeed a certificate $c \subseteq f$ of size s, that we can only do such a knocking-out of certificates s times.

We now claim that if there is a y-certificate $c \subseteq f$ of size f then the construction will terminate at a stage t where $t \leq s+1$. This will be sufficient to demonstrate claims (*) and (**) given above. For (*), note that if the construction terminates at all, it must be because case (1), (2) or (3) is satisfied at some stage of the construction. If there is a y-certificate $c \subseteq f$ for F, x with size bounded by s, then case (1) cannot hold. If case (2) holds it is clear that $g_{x,s}$ returns the correct answer. If case (3) holds it must be the case that c is among the remaining consistent certificates, and so $g_{x,s}$ returns the correct answer. For (**), we can see that at each stage of the construction the length of the dialogue is extended by at most s (since each certificate has size bounded by s). So the overall length of the dialogue between f and $g_{x,s}$ is bounded by $s \cdot (t-1) \leq s^2$.

It remains to demonstrate the claim. Suppose there is a y-certificate $c \subseteq f$ for F, x with size bounded by s. We first note that at each stage of the construction in which case 4 applies, it is never the case that the certificate chosen is a z certificate c' with $z > y$, since the witness c, which is certainly consistent with f, will appear before c' in the enumeration. Now suppose that the construction proceeds for s stages. We consider two cases.

Case I: c^s is a z-certificate with $z < y$. By Lemma 4.1, for each of $c^1, \ldots, c^s$ must disagree with c on some point of their domain, say x^i. Furthermore, each x^i must be unique (in order for c^i to be among the consistent certificates at stage i), so in fact $\{x^1, \ldots, x^s\} = \mathrm{dom}(c)$. Now by construction $x^1, \ldots, x^s \in \mathrm{dom}(f^s)$. So we must have $c \subseteq f^s$, and the construction terminates at stage $s+1$.

Case II: c^s is a y-certificate. Consider any remaining z-certificate c' with $z > y$. Again we unique have points $x^1, \ldots, x^s$ such that c' disagrees with c^i on x^i, and which in fact comprise $\mathrm{dom}(c')$, so that $\mathrm{dom}(c') \subseteq \mathrm{dom}(f^s)$. But then it must be the case that c' is not consistent with f^s. So the only remaining certificates consistent with f^s are y-certificates, and again the construction terminates at stage $s+1$. □

Kapron [1999] presents a class of functionals which are *feasibly continuous*. Such functionals have certificates whose size is *feasible*, or *polynomial*, in terms of the size of the inputs f and x. Of course, it is not entirely clear what it means to be polynomial in the size of a *function* f. One approach is to work backwards from the situation presented by type-one functions: a type-one function f has *polynomial growth rate* if for some polynomial p, $|f(x)| \leq p(|x|)$ for all x. Thus we propose that to be polynomial in f and x means to be bounded, for all f and x, by $|F(f,x)|$ where F is a functional with polynomial growth rate. Now it remains to characterize the functionals with polynomial growth rate. We begin with Mehlhorn's [1976] class of type-two, *polynomial time* functionals, which we will denote BFF (see [Cook and Kapron 1990] for an explanation of this terminology). The following presentation of BFF is based on that of Townsend [1990].

Definition 4.4. A rank (k, l) functional F is obtained by

—*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

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

In the case of limited recursion on notation, the F is defined only if the final inequality holds. Let $\{F_1,\dots,F_k\}$ be a set of functionals. The class $\mathrm{BFF}(\{F_1,\dots,F_k\})$ of functionals *basic feasible in* $\{F_1,\dots,F_k\}$ is the smallest class containing all poly-time (type-1) functions, $F_1,\dots,F_k$, and the *application functional* Ap defined 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 $\mathrm{BFF}(\emptyset)$.

Remark 4.5. 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.

Certainly, if all functionals in BFF are polynomial time computable, then they all have polynomial growth rate. However, BFF adds an extra restriction, namely computability, which is not essential. This restriction may be relaxed by allowing computation relative to a (type-one) oracle with polynomial growth rate. Thus we define the class $\mathbf{Poly}_1$ of second-order polynomial bounds to be all functionals of the form $\lambda f \lambda x.|F(f,x)|$, where F is a functional for which there exists a (type-one) function g with polynomial growth rate and a functional $G \in \mathrm{BFF}$ such that $F(f,x) = G(g,f,x)$ for all f and x.

Note that every functional in $\mathbf{Poly}_1$ is $\mathbf{Poly}_1$-sequential. This follows from the fact that sequential functionals are closed under composition and limited recursion on notation. In both cases, we need only to concatenate dialogues in the appropriate way to obtain the functional resulting from the closure condition. So we can apply Theorem 4.2 to obtain:

COROLLARY 4.6. **[Kapron 1999]** *Every functional which is* $\mathbf{Poly}_1$*-continuous is also* $\mathbf{Poly}_1$*-sequential.*

5. A LOWER BOUND

The $\mathbf{Poly}_1$-continuous functionals have the property that they depend on only a *feasible* (polynomially bounded) amount of information about their function inputs (relative to some understanding of what it means to be polynomially bounded in a *function*). It is possible to give a more direct definition of continuous functionals which depend on a polynomially bounded amount of information about their function inputs using *second-order polynomials*, and the notion of a *size* of a function.

Second-order polynomials were first introduced by Kapron and Cook [1996], who used them to give an alternate characterization of BFF.

Definition 5.1. A *second-order polynomial* over function variables $f_1, \ldots, f_k$ and number variables $x_1, \ldots, x_l$ is one of the following:

(1) A number constant $n \in \mathbb{N}$
(2) A number variable x_i
(3) $\mathbf{p} + \mathbf{q}$, where $\mathbf{p}$ and $\mathbf{q}$ are second-order polynomials
(4) $\mathbf{p} \cdot \mathbf{q}$, where $\mathbf{p}$ and $\mathbf{q}$ are second-order polynomials
(5) $f_i(\mathbf{p})$, where $\mathbf{p}$ is a second-order polynomial.

If $\mathbf{p}$ is a second-order polynomial, $f_1, \ldots, f_k : \mathbb{N} \to \mathbb{N}$ and $x_1, \ldots, x_l \in \mathbb{N}$, then $\mathbf{p}(f_1, \ldots, f_k, x_1, \ldots, x_l)$ denotes a value in $\mathbb{N}$, defined in the obvious way.

Defining the size of a function f is somewhat trickier. The basic idea is that we want a measure of the worst (largest) output that could be produced by f for inputs of a given size (see [Kapron and Cook 1996] for a discussion of this).

Definition 5.2. For any function $f : \mathbb{N} \to \mathbb{N}$, the *size* of f, denoted $|f|$, is the function $\lambda n. \max_{|y| \leq n} |f(y)|$.

The following result from [Kapron and Cook 1996] demonstrates the importance of second-order polynomials in the characterization of type-two polynomial time computability.

THEOREM 5.3. **[Kapron and Cook 1996]** *A functional is in* BFF *iff it is computed by an oracle Turing machine M such that for all inputs f and x, M's running time is bounded by a second-order polynomial in $|f|$ and $|x|$.*

Let $\mathbf{Poly}_2$ be the set of all functionals of the form $\lambda f \lambda x.\mathbf{p}(|f|, |x|)$, where $\mathbf{p}$ is a second-order polynomial.

It is not hard to see that there is no second-order polynomial $\mathbf{p}$ such that the functional $\lambda f \lambda x.|f|(|x|)$ is $\mathbf{p}$-continuous. If it were, there must be an ordinary polynomial p such that for any 0-1 valued f, the value of $|f|(|x|)$ is determined by a certificate of size $p(|x|)$. Choose an x_0 such that $2^{|x_0|} > p(|x_0|)$. Let $f_0 = \lambda x.0$. Since $|f_0|(|x_0|) = 0$, there must be a 0-certificate $c_0 \subseteq f_0$ for $\lambda f \lambda x.|f|(|x|)$ at x_0, such that $|c_0| \leq p(|x_0|)$. Now let g be any function such that $c_0 \subseteq g$ but $g(y) > 0$ for some y such that $|y| \leq |x_0|$. This is possible by the choice of x_0. But now $|g|(|x_0|) > |f_0|(|x_0|)$, contradicting the fact that c_0 is a certificate.

So we cannot apply Theorem 4.2 to obtain a result analogous to Corollary 4.6 for the $\mathbf{Poly}_2$-continuous functionals. In fact, there is a $\mathbf{Poly}_2$-continuous functional which is not $\mathbf{Poly}_2$-sequential. The remainder of this section is devoted to demonstrating the existence of such a functional.

The functional F_R defined below is closely related to a Boolean function defined in [Impagliazzo and Naor 1988], which is concerned with the power of Boolean decision trees for computing search functions.

The definition of F_R depends on a version of Ramsey's theorem (see, for example, [Graham et al. 1990]). F_R is defined in terms a a certain partial colouring function defined on complete graphs (denoted K_x):

Definition 5.4. For any function f and number $x > 0$, define the partial colouring function $\chi_{f,x}$ on K_x as follows: suppose that the edges of K_x are numbered

$1, 2, \ldots, \binom{x}{2}$. Then for $1 \leq i \leq \binom{x}{2}$

$$\chi_{f,x}(i) = \begin{cases} 0 & \text{if } f(i) < 2^x - 1 \text{ and even} \\ 1 & \text{if } f(i) < 2^x - 1 \text{ and odd} \end{cases}$$

Note that if $f(i) \geq 2^x$, then $\chi_{f,x}(i)$ is undefined. Given a $\chi_{f,x}$, we say that it has a *monochromatic* K_k if there is a set of k nodes such that $\chi_{f,x}$ colours the complete subgraph on these nodes all 0 or all 1.

The functional F_R ("Ramsey functional") is defined as follows: $F_R(f, 0) = 0$, and for $x > 0$,

$$F_R(f,x) = \begin{cases} 0 & \text{if } \chi_{f,x} \text{ has a monochromatic} \\ & \quad K_k \text{ with } k = \frac{1}{2}\log x; \\ 1 & \text{if } \chi_{f,x} \text{ has no monochromatic} \\ & \quad K_k \text{ with } k = \frac{1}{2}\log x. \end{cases}$$

THEOREM 5.5. *F_R is* $\mathbf{Poly}_2$*-continuous.*

PROOF: We will show that F_R is $\lambda f \lambda x.|x|^2(|f|(2|x|)^2 + |x|^2)$-continuous. First, suppose that $F_R(f,x) = 0$. Then there is a certificate, consisting of all the edges in the monochromatic subgraph. Such a certificate will have size bounded by $|x|^4$, since there will be $\binom{k}{2} \leq |x|^2$ values in the domain of the certificate (one for each edge of the monochromatic subgraph), and each value in the domain of the certificate is no greater than $\binom{x}{2}$. Now suppose that $F_R(f,x) = 1$. We now appeal to a result of Ramsey theory (see [Graham et al. 1990]) which states that any (total) 2-colouring of K_x has a monochromatic K_k where $k = \frac{1}{2}\log x$. Hence, if $F_R(f,x) = 1$, it follows that $\chi_{f,x}$ is undefined for some i, $1 \leq i \leq \binom{x}{2}$. Then it must be the case that $f(i) \geq 2^x - 1$, and so $|f(i)|^2 \geq x^2 \geq \binom{x}{2}$ and $|i| \leq 2|x|$. Thus there is a certificate, consisting of all the edges of K_x, of size $|x|^2|f|(2|x|)^2$. $\square$

Before proving that F_R is not $\mathbf{Poly}_2$-sequential, we need a simple result regarding the behaviour of sequential functionals.

PROPOSITION 5.6. *Let $F = F_g$ and $Q =$ Queries(g, f, x). Then for any f' that coincides with f on Q we have that g's dialogue with f is identical to its dialogue with f' (and hence* Queries$(g, f', x) = Q$*), and $F(f,x) = F(f',x)$.*

THEOREM 5.7. *F_R is not* $\mathbf{Poly}_2$*-sequential.*

PROOF: The proof is by contradiction. Suppose that $F_R = F_g$, and that there is a second-order polynomial $\mathbf{p}$ such that for every f and x, the dialogue between g_x and f has length bounded by $\mathbf{p}(|f|, |x|)$.

First note that there must be an ordinary polynomial p such that for any 0-1 valued f, $\mathbf{p}(|f|, |x|) \leq p(|x|)$. Next, by a lower bound on Ramsey numbers we have that for sufficiently large x, there is a 2-colouring χ_0 of the complete graph on $x^{0.25}$ nodes with no monochromatic K_k where $k = \frac{1}{2}\log x$ (see [Graham et al. 1990], Theorem 4.2.1).

Choose an x_0 for which such a χ_0 exists, and for which $\binom{x_0^{0.25}}{2} > p(|x_0|)$ (*). Note that for any 0-1 valued input function f, the number of elements of the set Queries(g, f, x_0) is at most $p(|x_0|)$ (**).

We will now construct functions f and f' such that:

(1) $\text{Queries}(g, f, x_0) = \text{Queries}(g, f', x_0)$
(2) $F_R(f, x_0) = 0$
(3) $F_R(f', x_0) = 1$

By Proposition 5.6, we immediately obtain a contradiction.

The definition of f is based on g and χ_0. Fix an enumeration of the edges of the complete graph on $x_0^{0.25}$ nodes. Let q_i denote the ith query made to f by g. Then $f(q_i) = \chi_0(i)$. For all other inputs x, $f(x) = 0$. To see that this definition makes sense, note that by (**), g can make at most $p(|x_0|)$ queries to f, and so by (*) f is well-defined. By the bound given in the proof of Theorem 5.5, χ_{f,x_0} must have a monochromatic K_k where $k = \frac{1}{2}\log x_0$, so that $F_R(f, x_0) = 0$.

The definition of f' is similar: $f'(q_i) = \chi_0(i)$, while $f'(x) = 2^{x_0}$ elsewhere. By appealing to Proposition 5.6 and the correctness of the definition of f, we have that f' is also well defined. Clearly, by construction we have $\text{Queries}(g, f, x_0) = \text{Queries}(g, f', x_0)$. However, by the choice of χ_0 we have $F_R(f', x_0) = 1$. □

Remark 5.8. We have in fact shown that there is an input f_0 such that, for sufficiently large x, any sequential strategy for computing $F_R(f_0, x)$ requires an exponential number (in $|x|$) of queries to f_0.

6. RELATED RESULTS

Townsend [1990] considers the relationship between $\mathbf{Poly}_1$-continuity and nondeterministic feasible computability relative to an oracle. 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 $G \in$ BFF 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 be stated using our terminology, with the following added definition:

Definition 6.1. A functional F is $\mathbf{Poly}_1$-*bounded* if there is a $G \in$ BFF such that for all f and x, $|F(f, x)| \le |G(f, x)|$

THEOREM 6.2. **[Townsend 1990]** *If F is $\mathbf{Poly}_1$-continuous functional which is also $\mathbf{Poly}_1$-bounded, then there is a function g such that F is feasibly nondeterministically computable in some g.*

By way of comparison, note that for any $\mathbf{Poly}_1$-bounded F, F is $\mathbf{Poly}_1$-sequential iff F is basic feasible (that is, feasibly *deterministically* computable) in some *poly-bounded g*. So it follows by Corollary 4.6 that if such a $\mathbf{Poly}_1$-bounded F is feasibly continuous then it is feasibly deterministically computable in some poly-bounded g. So we have obtained a considerable strengthening of Townsend's result. An interesting open question is whether or not a version of Townsend's result holds for $\mathbf{Poly}_2$.

All the results of this paper pertain to full collections of functionals at type level two. We have not addressed the issue of how effective versions of these collections are related. This question has been considered by Royer. He defines an effective version of the $\mathbf{Poly}_1$-continuous functionals, using the notion of a *poly-time associate*. The effective version of the $\mathbf{Poly}_1$-sequential functionals is just BFF itself. Royer shows that if the effective $\mathbf{Poly}_1$-continuous functionals coincide with BFF, then there is a polynomial time algorithm for factoring integers.

7. CONCLUSIONS AND FUTURE WORK

The relationship between continuity and sequentiality has long been an issue in the theory of computability for higher-type functions, and recent work has shed more light on the issue, in particular for total functionals. The work presented here formalizes the idea that, even in cases where classes of functionals coincide in terms of expressive power they may differ when considerations about efficiency are taken into account. We have shown that techniques from ordinary complexity theory may be applied fruitfully in this area.

This paper presents a starting-point for a number of possible threads of investigation. At type level two, a fuller investigation of the efficiency of various models of computation are possible. The relationship between bounded continuity, and computability with bounded nondeterminism should be more clearly delineated, perhaps in terms of a version of PCF++ with a bounded $\exists$ operator. Some of these issues will be dealt with in the full version of this paper. At higher types, we can investigate the recent results by Berger, Plotkin and Normann relating effective continuity and PCF definability with respect to efficiency. In another direction, it should be possible to develop some sort of complexity theory for computation over the sequential functionals of van Oosten and Longley. In short, recent developments in the theory of higher-type computability raise quite a number of interesting problems in the theory of higher-type complexity.

ACKNOWLEDGMENTS

We would like to thank Martin Hoffman, Russell Impagliazzo, Hanno Nickau, and Jim Royer for helpful discussions and suggestions.

REFERENCES

Berger, U. 1993. Total sets and objects in domain theory. *Annals of Pure and Applied Logic* 60, 91–117.

Blum, M. and Imagliazzo, R. 1987. Generic oracles and oracle classes. In *Proc. 28th Ann. IEEE Symposium on Foundations of Computer Science*. 118–126.

Cook, S. 1990. Computability and complexity of higher type functions. In *Proc. MSRI Workshop on Logic from Computer Science*, Y. Moschovakis, Ed. Springer-Verlag, 51–72.

Cook, S. and Kapron, B. 1990. Characterizations of the basic feasible functionals of finite type. In *Feasible Mathematics*, S. Buss and P. Scott, Eds. Birkhauser, Boston, MA, 71–93.

Gandy, R. and Hyland, M. 1977. Computable and recursively countable functions of higher type. In *Logic Colloquium '76*, R. Gandy and M. Hyland, Eds. North-Holland, Amsterdam, 407–438.

GRAHAM, R., ROTHSCHILD, B., AND SPENCER, J. 1990. *Ramsey Theory*, 2nd ed. John Wiley and Sons, Inc., New York.

HARTMANIS, J. AND HEMACHANDRA, L. 1990. Robust machines accept easy sets. *Theoretical Computer Science* 74, 217–225.

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

KAPRON, B. 1991. Feasible computation in higher types. Ph.D. thesis, University of Toronto, Department of Computer Science.

KAPRON, B. 1999. Feasibly continuous type-two functionals. *computational complexity* 8, 188–201.

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

KLEENE, S. 1959a. Countable functionals. In *Constructivity in Mathematics*, A. Heyting, Ed. North-Holland, Amsterdam, 81–100.

KLEENE, S. 1959b. Recursive functionals and quantifiers of finite types I. *Trans. Amer. Math. Soc.* 91, 1–52.

KLEENE, S. 1963. Recursive functionals and quantifiers of finite types II. *Trans. Amer. Math. Soc.* 108, 106–142.

KO, K.-I. 1986. Applying techniques of discrete complexity theory to numerical computation. In *Studies in Complexity Theory*, R. Book, Ed. Pitman Publishing Ltd., London, 1–62.

KREISEL, G. 1959. Interpretation of analysis by means of functionals of finite type. In *Constructivity in Mathematics*, A. Heyting, Ed. North-Holland, Amsterdam, 101–128.

LONGLEY, J. 1998. The sequentially realizable functionals. Tech. Rep. ECS-LFCS-98-402, University of Edinburgh LFCS. December.

LONGLEY, J. 2001. Notions of computability at higher types I. Available from `www.dcs.ed.ac.uk/home/jrl/notions1.ps`.

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

NORMANN, D. 1998. Computability over the partial continuous functionals. Pure Mathematics Preprint Series 1998 2, Department of Mathematics, University of Oslo. Revised version, October 1998.

VAN OOSTEN, J. 1999. A combinatory algebra for sequential functionals of finite type. In *Models and Computability: Invited Papers from Logic Colloquium '97*, S. Cooper and J. Truss, Eds. Cambridge University Press, Cambridge.

PLOTKIN, G. 1999. Full abstraction, totality and PCF. *Math. Struct. Comp. Sci.* 9, 1–20.

ROYER, J. On continuous type-2 feasible functionals. In preparation.

TAIT, W. 1962. Unpublished manuscript.

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

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

Received January 2001; revised August 2001; accepted December 2001