A New Characterization of Mehlhorn's Polynomial Time Functionals

(Extended Abstract) *

Bruce Kapron
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
bmkapron@cs.cmu.edu

Stephen A. Cook
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada, M5S 1A4
sacook@theory.toronto.edu

1 Introduction

A *type 1 function* is a total mapping from $\mathbf{N}$ to $\mathbf{N}$. We will denote the set of all such functions by ${}^{\mathbf{N}}\mathbf{N}$. A *type 2 functional* is a total mapping from $({}^{\mathbf{N}}\mathbf{N})^k \times \mathbf{N}^l$ to $\mathbf{N}$, for some k, l. More specifically, we will call a mapping of this sort a *functional with rank* (k, l).

For type 1 functions, there is a well established notion of computational feasibility. Namely a function is feasible if it is computable in polynomial time on a Turing machine. More specifically, a function f is poly time if there is a TM M and a polynomial p such that for all x, M with input x computes $f(x)$ and runs in time $p(n)$, where $n = |x|$, and for $x \in \mathbf{N}$, $|x|$ denotes the length of the binary notation of x, that is $\lceil \log(x+1) \rceil$. This notion of feasibility is robust; that is, it is independent of the computational model used, assuming that the model is 'reasonable'. In [1], Cobham presented a machine independent characterization of computational feasibility, via an inductive definition, that is, a definition which characterizes the feasible functions as the smallest class containing certain initial functions and closed under certain closure conditions. Cobham's definition, while important, lacks the intuitive appeal of the machine based characterization because intuitively, feasibility depends on a notion of bounding computational resources (in this case running time) in a general computational model in some natural way.

Questions about feasibility arise when dealing with type 2 functionals as well, for example, in the study of reducibilities ([9]), computable analysis ([7]), and descriptive set theory ([10]). Also, in [3] feasibility for functions of arbitrary finite type is presented as a means of interpreting systems of feasibly constructive arithmetic. Constable's paper [2] appears to be the first to consider the notion of feasibility for type 2 functionals. Mehlhorn's study [9] of feasible reducibilities proceeds from Constable's work. Here, a class of *poly time operators* is defined, using a generalization of Cobham's definition. Subsequent studies, such as [10], take Mehlhorn's approach. Cook and Urquhart [3] independently arrived at a characterization of type 2 feasibility which coincides with Mehlhorn's. This is a characterization based on terms in the equational logic PV^ω, which is used to interpret systems of feasibly constructive arithmetic. In [5], the authors study the computational properties of these terms. The work done to date in this area does not address the question of whether there is a natural machine based definition of Mehlhorn's class. In this paper, we provide an affirmative answer to the question. In order to do so, we need to define the notion 'poly time computable in the length of type 1 inputs'. We can then show that Mehlhorn's class is exactly the class of functionals so computable. The proof of this result is not a simple generalization of Cobham's equivalence proof; it depends on the fact that Mehlhorn's class is closed under an unusual form of simultaneous limited recursion on notation, and requires an analysis of the structure of oracle queries in time bounded computations.

Our model for type 2 computability is a generalization of the familiar multi-tape oracle Turing machine (OTM). Oracles now are functions, rather than sets. In addition to the normal work tapes, there is an *oracle query tape* and an *oracle answer tape* for each function input. These tapes are infinite in one direction. In order to query a function oracle at an input x, the machine writes x (in binary) on the corresponding query tape, moves the read head on the oracle tape to the beginning, and enters a query state for that oracle. In the next step the value of the function at the specified input is written (in binary) at the beginning of the corresponding answer tape, and the head of the answer tape is returned to the leftmost position. The rest of the answer tape is overwritten with blanks. There is also a special, read-only *input tape*. One work tape is specified as the *output tape*.

*The results presented here appear in detail in [8]

CH3062-7/91/0000/0342$01.00 © 1991 IEEE

An OTM M computes a functional F_M of rank (k,l) if it has k oracle query states, and for all $f_1,\ldots,f_k$ and $x_1,\ldots,x_l$, whenever M is started with $x_1,\ldots,x_l$ written in binary (and separated by blanks) on its input tape, and when f_i is the function associated with query state i, M halts with $F_M(\vec{f},\vec{x})$ written at the beginning of its output tape, followed by blanks and with the read head of the work tape in the leftmost position.

The running time of a machine for a particular input is the sum of the costs of the steps it performs before halting. Steps not involving oracle calls have unit cost. The cost of an oracle call is just the length of the value written on the answer tape for that call. In other words, the cost of calling oracle f with input x is $\max\{1,|f(x)|\}$. Note that this means that, in general, the running time of a machine will be greater than the number of steps it takes. We say that the running time of M is bounded by F if for all inputs $x_1,\ldots,x_l$ and $f_1,\ldots,f_k$, the running time of M is bounded by $F(\vec{f},\vec{x})$.

We now consider an example to illustrate the basic issue involved in defining feasibility for type 2 functionals, namely accounting for the growth of function inputs. Recall that for sets $A,B\subseteq\mathbf{N}$, A is *poly time Cook reducible to* B ($A\leq_C^p B$) if there is an OTM M an a polynomial p such that for all x, M^B with input x runs in time $p(|x|)$ and accepts x iff $x\in A$. Suppose we want to generalize this definition to arbitrary functions. We then would have $f\leq_C^p g$ iff there is an OTM M and a polynomial p so that for all x, M with oracle g and input x runs in time $p(|x|)$ and returns $f(x)$. With this definition, we lose reflexivity. That is, there are functions f, for example $f(x)=2^x$, for which $f\not\leq_C^p f$. This is because our definition of poly time reducibility did not account for the growth of f.

2 Basic feasible functionals

Cobham [1] gave an inductive definition of type 1 feasible functions, in terms of certain initial functions and closure conditions. The most important aspect of this definition is closure under *limited recursion on notation*. These feasible functions coincide exactly with the familiar poly time functions. Mehlhorn [9] generalized Cobham's definition to type 2 functionals, to define the class of *polynomial time operators*. We will consider a functional version of this generalization, based on that given by Townsend [10]. Note that we will refer to functionals in this class as *basic feasible functionals* (BFF's) rather than poly time functionals. This terminology is introduced in [5]. It is based on the fact that there are functionals which meet intuitive necessary conditions for feasibility, but are not BFF's [6]. in [5]. We first introduce some schemes for defining functionals.

Definition 2.1 F is defined from $H,G_1,\ldots,G_l$ by *functional composition* if for all $\vec{f},\vec{x}$,

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

F is defined from G by *expansion* if for all $\vec{f},\vec{g},\vec{x},\vec{y}$,

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

F is defined from G,H,K by *limited recursion on notation* (LRN) if for all $\vec{f},\vec{x},y$,

$$\begin{aligned} F(\vec{f},\vec{x},0) &= G(\vec{f},\vec{x}) \\ F(\vec{f},\vec{x},y) &= H(\vec{f},\vec{x},y,F(\vec{f},\vec{x},\lfloor\tfrac{y}{2}\rfloor)), \qquad y>0 \\ |F(\vec{f},\vec{x},y)| &\leq |K(\vec{f},\vec{x},y)|. \end{aligned}$$

Definition 2.2 Let X be a class of type 2 functionals. The class of *basic feasible functionals defined from* X ($\mathrm{BFF}(X)$) is the smallest class of functionals containing X, all type 1 poly time functions and the application functional Ap, defined by $Ap(f,x)=f(x)$, and which is closed under functional composition, expansion, and limited recursion on notation. If $F\in\mathrm{BFF}(X)$ we say that F is *basic feasible in* X. The *basic feasible functionals* (BFF's) are just $\mathrm{BFF}(\emptyset)$.

Note that if we define poly time reducibility by: $f\leq_C^p g$ iff there is a BFF F such that for all x, $f(x)=F(g,x)$, then $\leq_C^p$ is reflexive and transitive.

The BFF's have a strong closure property with respect to computation by OTM's. We now define this property.

Definition 2.3 A class X of functionals has the *Ritchie-Cobham property* if for all F, $F\in X$ iff there is an OTM M and some $G\in X$ so that M computes F and for all inputs $\vec{f},\vec{x}$, the running time of M is bounded by $|G(\vec{f},\vec{x})|$.

Theorem 2.4 (Mehlhorn [9]) The BFF's have the Ritchie-Cobham property.

In fact, Mehlhorn proves this result for a slightly different model. In this model, an oracle call has unit cost no matter what value is returned. However, it is not hard to modify Mehlhorn's proof to obtain the same result for our model. A proof for our model is given in [5], in the more general setting of functionals over all finite types.

This result provides some evidence of the naturalness of the BFF's. However, it is not a truly dynamic

characterization of this class, that is, it still depends on the inductive definition. We will now show that such a dynamic characterization is indeed possible.

In order to do so, we first need to show that the BFF's are closed under a scheme of recursion which seems to be quite powerful. We begin by considering a simple extension of LRN. Let $\vec{F}(\vec{f},\vec{x})$ denote $F_1(\vec{f},\vec{x}),\ldots,F_k(\vec{f},\vec{x})$.

Definition 2.5 $F_1,\ldots,F_k$ are defined from $\vec{G},\vec{H},\vec{K}$ by *simultaneous limited recursion on notation* (SLRN) if for all $\vec{f},\vec{x},y$, and for $1\le i\le k$,

$$F_i(\vec{f},\vec{x},0) = G_i(\vec{f},\vec{x}) \quad (2a)$$

$$F_i(\vec{f},\vec{x},y) = H_i(\vec{f},\vec{x},y,\vec{F}(\vec{f},\vec{x},\lfloor\tfrac{y}{2}\rfloor)), \quad y>0 \quad (2b)$$

$$|F_i(\vec{f},\vec{x},y)| \le |K_i(\vec{f},\vec{x},y)|. \quad (2c)$$

Lemma 2.6 *If $F_1,\ldots,F_k$ are defined from $\vec{G},\vec{H},\vec{K}$ by SLRN, then F_i is basic feasible in $\vec{G},\vec{H},\vec{K}$, $1\le i\le k$.*

PROOF. Recall that for $k\in\mathbf{N}$, there are poly time functions $\lambda x_1\ldots x_k.\langle x_1,\ldots,x_k\rangle$ and Π_i^k, $1\le i\le k$, such that $\Pi_i^k(\langle x_1,\ldots,x_k\rangle)=x_i$. Now it is not hard to see that we can define, using LRN, a functional F, basic feasible in $\vec{G},\vec{H},\vec{K}$, such that $F_i(\vec{f},\vec{x},y)=\Pi_i^k(F(\vec{f},\vec{x},y))$. ∎

SLRN is the "standard" form of simultaneous limited recursion on notation. We now consider a more powerful type of simultaneous recursion on notation.

Definition 2.7 $F_1,\ldots,F_k$ are defined from $\vec{G},\vec{H},\vec{K}$ by *multiple limited recursion on notation* (MLRN) if for all $\vec{f},\vec{x},y$, 2a and 2b above hold, along with the following bounding conditions, in place of 2c:

$$|F_1(\vec{f},\vec{x},y)| \le |K_1(\vec{f},\vec{x},y)|$$

$$|F_i(\vec{f},\vec{x},y)| \le |K_i(\vec{f},\vec{x},y,\vec{F}_{i-1}(\vec{f},\vec{x},y))|, \quad 2\le i\le k,$$

where $\vec{F}_{i-1}(\vec{f},\vec{x},y)$ denotes the sequence

$$F_1(\vec{f},\vec{x},y),\ldots,F_{i-1}(\vec{f},\vec{x},y).$$

MLRN is a generalization of a scheme introduced by Cook in [4]. The apparent power of this scheme compared to SLRN arises from the use of weaker bounds for $|F_i(\vec{f},\vec{x},y)|$. Otherwise, MLRN is identical to SLRN. The following result shows that, in fact, the BFF's can capture MLRN.

Theorem 2.8 *If $F_1,\ldots,F_k$ are defined by MLRN from $\vec{G},\vec{H},\vec{K}$ then F_i is basic feasible in $\vec{G},\vec{H},\vec{K}$, $1\le i\le k$.*

PROOF. We only consider the case where $k=2$ (the general case requires an argument by induction on k.) We use the notation $v\subseteq y$ to indicate that the binary notation for v is an initial segment of the binary notation for y. To begin, it is not hard to modify the definition by MLRN of F_1,F_2 to obtain functionals E_1,E_2, basic feasible in G_i,F_i,K_i, $i=1,2$, so that for all z and y', if

$$|F_2(\vec{f},\vec{x},v)| \le |K_2(\vec{f},\vec{x},y',z)|$$

whenever $v\subseteq y$, then

$$E_i(\vec{f},\vec{x},y',z,y) = F_i(\vec{f},\vec{x},y), \qquad i=1,2. \quad (2d)$$

We will now show that there is a functional P, basic feasible in G_i,H_i,K_i, $i=1,2$, so that for all y and all $v\subseteq y$,

$$|F_2(\vec{f},\vec{x},v)| \le |K_2(\vec{f},\vec{x},P_1(\vec{f},\vec{x},y),P_2(\vec{f},\vec{x},y))|. \quad (2e)$$

where $P_i(\vec{f},\vec{x},y)=\Pi_i(P(\vec{f},\vec{x},y))$, $i=1,2$. By 2d we will then have, for $i=1,2$,

$$F_i(\vec{f},\vec{x},y) = E_i(\vec{f},\vec{x},P_1(\vec{f},\vec{x},y),P_2(\vec{f},\vec{x},y),y),$$

so that F_1 and F_2 are basic feasible in G_i,H_i,K_i, $i=1,2$. We will define P so that it satisfies

$$P(\vec{f},\vec{x},y) = \langle v, F_1(\vec{f},\vec{x},v)\rangle, \quad (2f)$$

where $v\subseteq y$ maximizes $|K_2(\vec{f},\vec{x},v,F_1(\vec{f},\vec{x},v))|$. It is then clear that P satisfies 2e for all $v\subseteq y$. We define P as follows: $P(\vec{f},\vec{x},0)=\langle 0,G_1(\vec{f},\vec{x})\rangle$, while for $y>0$, $P(\vec{f},\vec{x},y)=\langle y,z\rangle$ if $|K_2(\vec{f},\vec{x},y,z)|\ge|K_2(\vec{f},\vec{x},t_1,t_2)|$, and $P(\vec{f},\vec{x},y)=t$ otherwise, where $t=P(\vec{f},\vec{x},\lfloor\frac{y}{2}\rfloor)$, $t_i=\Pi_i(t)$, $i=1,2$, and

$$z = H_1(\vec{f},\vec{x},y,\vec{E}(\vec{f},\vec{x},t_1,t_2,\lfloor\tfrac{y}{2}\rfloor)).$$

We show that P satisfies 2f for all y, by induction on the notation of y. Note that this also means that for all y,

$$|P(\vec{f},\vec{x},y)| \le |\langle y, \max_{v\subseteq y} K_1(\vec{f},\vec{x},v)\rangle|,$$

so that P is definable by LRN from functionals basic feasible in G_i,H_i,K_i, $i=1,2$. ∎

3 Basic poly time functionals

In order to generalize the definition of poly time to type 2 functionals, we need to extend the notion of a polynomial to include variables for functions, and we need to define a notion of length for functions.

Definition 3.1 *First-order variables* are elements of the set $\{n_1, n_2, \ldots\}$. *Second-order variables* are elements of the set $\{L_1, L_2, \ldots\}$. *Second-order polynomials* are defined inductively:

- Any $c \in \mathbf{N}$ is a second-order polynomial
- First-order variables are second-order polynomials
- If P, Q are second-order polynomials and L is a second-order variable, then $P+Q$, $P \cdot Q$ and $L(P)$ are second-order polynomials.

We will refer to second-order polynomials as polynomials when the context makes this distinction clear. Suppose P is a polynomial, all of whose first-order variables are among $n_1, \ldots, n_s$ and all of whose second-order variables are among $L_1, \ldots, L_t$. Then for any sequence $f_1, \ldots, f_t$ of functions, and any sequence $x_1, \ldots, x_s$ of numbers, $P(\vec{f}, \vec{x})$, *P evaluated at* $\vec{f}, \vec{x}$ denotes some natural number. For example, if

$$P_0 = L_1(L_1(n_1 \cdot n_1)) + L_1(L_1(n_1) \cdot L_1(n_1)) + L_1(n_1) \tag{3a}$$

and $f(x) = x^2$, then

$$\begin{aligned} P_0(f,2) &= f(f(2 \cdot 2)) + f(f(2) \cdot f(2)) + f(2) \\ &= (4^2)^2 + (2^2 \cdot 2^2)^2 + 2^2 \\ &= 516. \end{aligned}$$

We now introduce a notion of length for functions. Our definition is motivated by two factors. First of all, we want to express the running time of a functional as a polynomial in the lengths of its function and number inputs. Hence the length of a function will expect lengths of numbers for its inputs. Secondly, we are often concerned with bounding running times, rather than determining them exactly. This is much easier if our bounding functions are monotone.

Definition 3.2 If $f : \mathbf{N} \to \mathbf{N}$ is a function, then $|f|$, *the length of* f, is the function $\lambda n. \max_{|y| \leq n} |f(y)|$.

We now introduce a type 2 analog for poly time functions, based on our generalizations of polynomials and lengths for functions.

Definition 3.3 A functional F is *basic poly time* if there is an OTM M and a second order polynomial P such that for all inputs $\vec{f}, \vec{x}$, the machine M computes $F(\vec{f}, \vec{x})$, and has running time bounded by $P(|f_1|, \ldots, |f_k|, |x_1|, \ldots, |x_l|)$.

Theorem 3.4 *Every BFF is basic poly time.*

PROOF. By theorem 2.4, it suffices to show that if F is a BFF, then there is a polynomial P so that for all $\vec{f}, \vec{x}$, $|F(\vec{f}, \vec{x})| \leq P(|f_1|, \ldots, |f_k|, |x_1|, \ldots, |x_l|)$. We proceed by induction on the definition of F. The only nontrivial case is when F is defined by functional composition. Here we use the fact that the function $\lambda \vec{x}.P(|f_1|, \ldots, |f_k|, |x_1|, \ldots, |x_l|)$ is monotone nondecreasing. ∎

Surprisingly, the converse of 3.4 is also true, so that the BFF's and the basic poly time functionals coincide. The following example illustrates some of the difficulties involved in proving the converse.

Let F_1 be the function such that $F_1(f, x) = (\mu k < x)(\max_{i \leq k} |f(i)| = k)$ if such a k exists, and $F_1(f, x) = x$ otherwise. It is easy to see that this functional is basic poly time. For inputs f, x we can compute F_1 in time bounded by $[|f|(|x|)]^2$ as follows: just evaluate f at successive inputs, starting with 0, until we find a point k such that $F_1(f, x) = k$ or reach x. Now we will make at most $F_1(f, x) + 1$ such evaluations, and each evaluation returns a value bounded by $|f|(|x|)$. The approximate run time bound is then obtained by noting that $F_1(f, x) \leq |f|(|x|)$. This approach to computing F_1 does not allow us to conclude that F_1 is a BFF. In particular, it appears that with such an approach, certain inputs f and x would require a recursion with a number of iterations exponential in $|x|$. However, this problem can be avoided with a nested recursion, as we will now show. In order to do so, we need to consider the auxiliary function F_2 such that $F_2(f, x)$ returns the smallest point y, $0 \leq y \leq F_1(f, x)$, such that y maximizes f over $\{0, \ldots, F_1(f, x)\}$, if such a y exists, and returns x otherwise. So if $F_1(f, x) < x$, $|f(F_2(f, x))| = F_1(f, x)$. Otherwise, $|f(F_2(f, x))| \geq x$. Let

$$F(f, x) = \langle F_1(f, x), F_2(f, x) \rangle,$$

and let

$$G(f, x, y) = F(f, \min(|x|, y)).$$

Clearly G is basic feasible (we can use LRN on x to do a "brute force" search). We define F using LRN, as follows:

$$\begin{cases} F(f, 0) = \langle 0, 0 \rangle \\ F(f, x) = \textbf{if } F_1(f, \lfloor \frac{x}{2} \rfloor) < \lfloor \frac{x}{2} \rfloor \textbf{ then } F(f, \lfloor \frac{x}{2} \rfloor) \\ \qquad\qquad \textbf{else } G(f, f(F_2(f, \lfloor \frac{x}{2} \rfloor))\#2, x) \\ |F(f, x)| \leq |\langle x, x \rangle|, \end{cases}$$

where $\#$ is the poly time function defined by $x \# y = 2^{|x| \cdot |y|}$. So $F_1(f, x) = \Pi_1(F(f, x))$.

To simplify the proof of the converse of 3.4, we will restrict our attention to functionals of rank $(1, 1)$. Basically, we want to show that if F is computed by an OTM M so that for all inputs f, x the running time

of M is bounded by $P(|f|,|x|)$, then there is a BFF G so that for all f,x the running time of M is bounded by $|G(f,x)|$. By theorem 2.4 we will then have that F is a BFF. More formally, we define a BFF Run_M, such that for any inputs f,x and $T \in \mathbf{N}$, $Run_M(f,x,T)$ returns an encoding of the computation of M with inputs f,x for $S(|T|)$ steps, where $S(|T|)$ denotes the least s such that the running time of M on inputs f,x after s steps exceeds $|T|$ (recall that the running time of M is the sum of the cost of each step, so that it is possible that the running time after $S(|T|)$ steps may exceed $|T|$. In particular, this can happen when the last step M performs is an oracle call.) Having defined this functional, our goal is to find a BFF G so that $F(f,x) = Run_M(f,x,G(f,x))$. Now if there were a BFF H such that $|H(f,x)| = |f|(|x|)$, our task would be trivial, since we could then obtain the BFF G from H. However, there is no such BFF H: if there were, then for a 0-1 valued input f, $H(f,x)$ could be computed in time polynomial in $|x|$. But this contradicts the fact that the number of queries required to compute $H(f,x)$ could be exponential in $|x|$.

Our goal now is to try to simplify P in such a way that the value of $P(|f|,|x|)$ can be feasibly computed without using a functional such as H. We begin by noting that the run time of M must be bounded in terms of values $f(y)$ such that M actually queries f at y. We formalize this notion as follows.

Definition 3.5 For any function f, any OTM M and any $t,x \in \mathbf{N}$, let $\mathcal{Q} = \mathcal{Q}(M,f,x,t)$ denote the *query set* consisting of all y such that M with inputs f,x queries f at y within the first $S(t)$ steps of its execution, where $S(t)$ is the least number of steps so that if M runs for $S(t)$ steps then its running time is at least t. For any set $\mathcal{Q} \subseteq \mathbf{N}$ and any function f, let $f_{\mathcal{Q}}$, the *query restriction of* f, be the function such that $f_{\mathcal{Q}}(y) = f(y)$ for all $y \in \mathcal{Q}$, and $f_{\mathcal{Q}}(y) = 0$ otherwise.

The important fact about $f_{\mathcal{Q}}$, $\mathcal{Q} = \mathcal{Q}(M,f,x,t)$, is that the behavior of M with inputs $f_{\mathcal{Q}},x$ and its behavior with inputs f,x are identical within the first $S(t)$ steps of its execution. In particular, we have for all t, if $t \geq P(|f_{\mathcal{Q}}|,|x|)$ then M must halt in $S(t)$ steps.

We will now use the above stated property of $f_{\mathcal{Q}}$ in order to simplify the polynomial which bounds the running time of M. We will use the polynomial P_0 given in 3a as a running example, rather than considering an arbitrary polynomial. If P_0 bounds the running time of M and $\mathcal{Q} = \mathcal{Q}(M,f,x,t)$, then there is a point $q_1 \in \mathcal{Q}$ such that

$$|q_1| \leq \max\{|x|,|x|^2\} \tag{3b}$$

and

$$|f(q_1)| \geq \max\{|f_{\mathcal{Q}}|(|x|),|f_{\mathcal{Q}}|(|x|^2)\}. \tag{3c}$$

That is, q_1 is a point which maximizes the *depth 1* subpolynomials of P_0, where the depth of a polynomial is just the maximum nesting of function calls. Likewise, there is a point $q_2 \in \mathcal{Q}$ so that

$$|q_2| \leq \max\{|f_{\mathcal{Q}}|(|x|^2),[|f_{\mathcal{Q}}|(|x|)]^2\} \tag{3d}$$

and

$$|f(q_2)| \geq \max\{|f_{\mathcal{Q}}|(|f_{\mathcal{Q}}|(|x|^2)),|f_{\mathcal{Q}}|([|f_{\mathcal{Q}}|(|x|)]^2)\}. \tag{3e}$$

So q_2 maximizes the depth 2 polynomials of P_0.

It is now easy to see that there is a first-order polynomial p_0 so that for any f,x, if q_1,q_2 are chosen satisfying 3b,3c, 3d,3e for $\mathcal{Q} = \mathcal{Q}(M,f,x,t)$, then

$$P_0(|f_{\mathcal{Q}}|,|x|) \leq p_0(|f(q_1)|,|f(q_2)|,|x|).$$

But then there is a BFF G_0 so that

$$P_0(|f_{\mathcal{Q}}|,|x|) \leq |G_0(f,q_1,q_2,x)|. \tag{3f}$$

Thus, we have reduced the problem of finding a basic feasible bounding functional to finding BFF's q_1^M, q_2^M so that $q_1^M(f,x)$ satisfies 3b and 3c, and $q_2^M(f,x)$ satisfies 3d and 3e, for $\mathcal{Q} = \mathcal{Q}(M,f,x,P_0(|f|,|x|))$. As a first step towards showing that there are such BFF's, we will consider q_i^M, $i = 1,2$, as a functional with an extra input r. This inputs bounds the number of distinct points at which M is allowed to query its function input. More formally, for $r \in \mathbf{N}$, let t_r be the least t such that $|\mathcal{Q}(M,f,x,t)| = r$, and let $q_i^M(f,x,r) = q_i$, $i = 1,2$, where q_1 satisfies 3b and 3c, and q_2 satisfies 3d and 3e, for $\mathcal{Q} = \mathcal{Q}(M,f,x,t_r)$.

We will show that q_i^M, $i = 1,2$, can be defined simultaneously by induction on the notation of r, using only BFF's. Now it follows from 3b that if $h(x) = 2^{|x|^2}$, then for all f,x, $|q_1^M(f,x,r)| \leq |h(x)|$. Likewise, by 3d it follows that there is a BFF H so that for all f,x, $|q_2^M(f,x,r)| \leq |H(f,q_1^M(f,x,r),x)|$. With these bounds we can conclude that q_1^M, q_2^M are definable by MLRN, using only BFF's, and so they are basic feasible. It remains to give the simultaneous definition of q_1^M, q_2^M. We first note that if r is presented in unary, then the definition would be quite straightforward. That is, the functionals $\mathbf{q}_i^M$, $i = 1,2$, defined by $\mathbf{q}_i^M(f,x,R) = q_i^M(f,x,|R|)$ are easily defined by MLRN. Now suppose that we have $q_i^M(f,x,\lfloor \frac{r}{2} \rfloor)$, $i = 1,2$. We can then obtain $q_i^M(f,x,r)$, $i = 1,2$, as follows: if M halts before making $\lfloor \frac{r}{2} \rfloor$ queries, we're done. Otherwise, let $t = t_{\lfloor \frac{r}{2} \rfloor}$ and $\mathcal{Q} = \mathcal{Q}(M,f,x,t)$. Now we know that $P(|f_{\mathcal{Q}}|,|x|) \geq t$. Moreover, $t \geq \lfloor \frac{r}{2} \rfloor$, since each query

has cost at least 1. But then, using the BFF G_0 which satisfies 3f, we can define a BFF G such that

$$|G(f, q_1^M(f,x,\lfloor \tfrac{r}{2} \rfloor), q_2^M(f,x,\lfloor \tfrac{r}{2} \rfloor), x)| = r.$$

But then $q_i^M(f,x,r)$ is equal to

$$q_i^M(f,x,G(f,q_1^M(f,x,\lfloor \tfrac{r}{2} \rfloor), q_2^M(f,x,\lfloor \tfrac{r}{2} \rfloor), x)).$$

We now want to eliminate the use of the parameter r. If we could show that there is a BFF R such that for all f,x, $R(f,x) \geq |Q(M,f,x,P(|f|,|x|))|$, we would be finished. What we will actually show is that there are r_1, r_2 which "approximate" $R(f,x)$, such that r_1 is basic feasible in f,x and r_2 is basic feasible in r_1, f, x, and such that a basic feasible bounding functional G can be obtained from the r_i's. Intuitively, r_i, $i = 1,2$, is $R(f,x)$ assuming that M queries f only at points y such that $|y| \leq |q_i|$. We first note that if $|Q(M,f,x,t)| \geq r$, then $t \geq r$. Now $P(|f_Q|,|x|) \geq t$. So if

$$T = G_0(f, q_1^M(f,x,r), q_2^M(f,x,r), x), \qquad (3g)$$

then $|T| \geq P(|f_Q|,|x|)$, so either M halts within $S(|T|)$ steps, or $|T| \geq r$. Now let A_{lmax} be a functional so that for all f,x,

$$|f(A_{lmax}(f,x))| = \max_{y \leq |x|} |f(y)|.$$

It is not hard to see that there is BFF A_{lmax} with this property. But $|f(A_{lmax}(f, 2\#T))| \geq |f|(|r|)$. In other words, if M on inputs f,x runs for long enough to make r queries, then we can feasibly compute $|f|(|r|)$ from f,x and r. We will now apply this for the bounding polynomial P_0. For inputs f,x, we will begin by trying to find q_1. For any Q, $|q_1| \leq |x|^2 \leq |x\#x|$. So we begin by setting $r_1 = x\#x$. Let T_1 be T as defined in 3g, for $r = r_1$. Now if M halts before making r_1 queries, it halts in $S(|T_1|)$ steps, so we don't need to go any further, since $|T_1|$ will bound the running time of M. Otherwise, we have a value $l_1 = A_{lmax}(f, 2\#T_1)$ so that $|f(l_1)| \geq |f(q_1)|$. Since $|q_2| \leq |f(q_1)|^2$, we now try $r_2 = f(l_1)\#f(l_1)$, and let T_2 be T as defined by 3g for $r = r_2$. Again, if M halts in $S(|T_2|)$ steps, we're done. Otherwise, we have a value $l_2 = A_{lmax}(f, 2\#T_2)$ so that $|f(l_2)| \geq |f(q_2)|$. Under the assumption that for inputs f,x, M does not halt in $S(|T_1|)$ or $S(|T_2|)$ steps, the running time of M must be bounded by $2 \cdot |f(l_2)| + |f(l_1)|$. So if

$$G(f,x) = \max\{T_1, T_2, Pad(2\#Ap(f,l_2), Ap(f,l_1))\},$$

then $F(f,x) = Run_M(f,x,G(f,x))$. Formalizing this argument for arbitrary bounding polynomials, we obtain

Theorem 3.6 *If F is a rank $(1,1)$ basic poly time functional, then F is basic feasible.*

4 Conclusions

We have given a natural machine based characterization of the basic feasible functionals. We believe that this new characterization will lead to a better understanding of the nature of resource-bounded computation in higher types, and will provide a more natural setting for applications.

5 Acknowledgements

We would like to thank Stephen Bellantoni for helping to formulate the definition of second-order bounding polynomials.

References

[1] Alan Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel ed., *Proc. of the 1964 Int. Cong. for Logic, Methodology and the Philosophy of Science*, pp. 24-30. North Holland, Amsterdam, 1964.

[2] Robert Constable. Type two computational complexity, *Proc. 5th ACM STOC* (1973) 108-121.

[3] Stephen Cook and Alasdair Urquhart. Functional interpretations of feasibly constructive arithmetic, *Proc. 21st STOC* (1989) 107-112.

[4] Stephen Cook. Iterated recursion is PV^ω-definable. Manuscript. November, 1989.

[5] Stephen Cook and Bruce Kapron. Characterizations of the basic feasible functionals of finite type. In S. Buss and P. Scott, eds., *Feasible Mathematics*, pp. 71-95. Birkhauser, 1990.

[6] Stephen Cook. Computability and complexity of higher type functions. To appear in *Proc. MSRI Logic Workshop*. 1990.

[7] H. Friedman and K. Ko. Computational Complexity of Real Functions. *Theoretical Computer Science* **20** (1982), 323-352.

[8] Bruce Kapron. *Feasible Computation in Higher Types*. PhD. Thesis, Department of Computer Science, University of Toronto, 1991.

[9] Kurt Mehlhorn. Polynomial and abstract subrecursive classes, *JCSS* **12** (1976) 147-178.

[10] Mike Townsend. Complexity for type 2 relations. *Notre Dame Journal of Formal Logic* **31** (1990), 241-262.