

# Some Applications of Logic to Feasibility in Higher Types

Aleksandar Ignjatovic and Arun Sharma

November 2, 2018

## Abstract

While it is commonly accepted that computability on a Turing machine in polynomial time represents a correct formalisation of the notion of a *feasibly computable* function, there is no similar agreement on how to extend this notion on *functionals*, i.e., what functionals should be considered feasible. One possible paradigm was introduced by K. Mehlhorn in [9], who extended Cobham’s definition of feasible functions to type 2 functionals. Subsequently, this class of functionals (with inessential changes of the definition) was studied by Townsend ([13]) who calls this class **POLY**, and by Cook and Kapron ([5]) who call the same class *basic feasible functionals*. Cook and Kapron gave an oracle Turing machine model characterization of this class. In this paper we demonstrate that the class of basic feasible functionals has recursion theoretic properties which naturally generalize the corresponding properties of the class of feasible functions, thus giving further evidence that the notion of feasibility of functionals mentioned above is correctly chosen. We also improve the Kapron-Cook result on machine representation.

Our proofs are based on essential applications of logic. We introduce a weak fragment of second order arithmetic with second order variables ranging over functions from  $\mathbb{N}^{\mathbb{N}}$  which suitably characterizes basic feasible functionals, and show that it is a useful tool for investigating the properties of basic feasible functionals. In particular we provide an example how one can extract feasible “programs” from mathematical proofs which use non-feasible functionals.

## 1 Introduction

### 1.1 Recursion Theory of Feasible Functionals

In [4] Cobham has established the following fundamental result: functions computable on a Turing machine in polynomial time are exactly functions which

can be obtained from the basic functions<sup>1</sup>  $\mathbf{o}(x) = 0, s_0(x) = 2x, s_1(x) = 2x + 1, i_k^n(x_1, \dots, x_n) = x_k$  and  $x \# y$  by composition and by limited recursion on notation:

$$f(\vec{x}, 0) = g(\vec{x}) \quad (1)$$

$$f(\vec{x}, 2y) = h_1(\vec{x}, y, f(\vec{x}, y)), \quad y > 0 \quad (2)$$

$$f(\vec{x}, 2y + 1) = h_2(\vec{x}, y, f(\vec{x}, y)) \quad (3)$$

$$|f(\vec{x}, y)| \leq |k(\vec{x}, y)| \quad (4)$$

It is easy to see that the same class of functions is obtained if we replace the schema of limited recursion on notation by a schema which instead of (2) and (3) has

$$f(\vec{x}, y) = h(\vec{x}, y, f(\vec{x}, \lfloor \frac{y}{2} \rfloor)), \quad y > 0 \quad (5)$$

Also, instead of condition (4), we can take the condition

$$|f(\vec{x}, y)| \leq q(|\vec{x}|, |y|) \quad (6)$$

where  $q$  is a polynomial with natural coefficients.

Finally, the same set can be obtained if the schema of limited recursion on notation is replaced by the following version of the schema of primitive recursion, in which  $p$  and  $q$  are polynomials and  $f^*$  is an auxiliary primitive recursive function.

$$f^*(\vec{y}, 0) = g(\vec{y}) \quad (7)$$

$$f^*(\vec{y}, x + 1) = h(\vec{y}, x, f^*(\vec{y}, x)) \quad (8)$$

$$(\forall x \leq p(|\vec{y}|)) (|f^*(\vec{y}, x)| \leq q(|\vec{y}|)) \quad (9)$$

$$f(\vec{y}) = f^*(\vec{y}, p(|\vec{y}|)) \quad (10)$$

From the foundational point of view we note that conditions (6) and (9) limit the growth rate of the function being defined using functions which are particularly simple to compute (polynomials), in the sense that their definition does not involve any recursion.

**Definition 1** A function  $F : (\mathbb{N}^{\mathbb{N}})^k \times \mathbb{N}^l \rightarrow \mathbb{N}$  is called a *type 2 functional of rank  $(k, l)$* .

One of our aims is to find analogues of the above schemata for a class of functionals which can be seen as natural extension of the class of feasible functions.

**Definition 2** A functional of rank  $(k, l)$  is obtained by

---

<sup>1</sup>Recall that  $|x|$  denotes the length of the binary representation of  $x$ , that is to say  $\lceil (\log_2 x) + 1 \rceil$ , with  $|0| = 0$ ;  $\lfloor \frac{1}{2}x \rfloor$  denotes the greatest integer less than or equal to  $x/2$ ;  $x \# y$  is equal to  $2^{|x| \cdot |y|}$ .

*Functional composition* from  $H(\vec{f}, \vec{y}, \vec{x}), G_1(\vec{f}, \vec{x}), \dots, G_l(\vec{f}, \vec{x})$  if

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

*Expansion* from  $G(\vec{f}, \vec{x})$  if

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

*Functional substitution* from  $H(\vec{g}, \vec{f}, \vec{x}), G_1(\vec{f}, \vec{x}, y), \dots, G_l(\vec{f}, \vec{x}, y)$  if

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

*Limited recursion on notation* from  $G, H_1, H_2, K$  if the following hold for all  $\vec{f} \in (\mathbb{N}^{\mathbb{N}})^k$  and all  $\vec{x} \in \mathbb{N}^l$ :

$$F(\vec{f}, \vec{x}, 0) = G(\vec{f}, \vec{x}) \quad (11)$$

$$F(\vec{f}, \vec{x}, 2y) = H_1(\vec{f}, \vec{x}, y, F(\vec{f}, \vec{x}, y)), \quad y > 0 \quad (12)$$

$$F(\vec{f}, \vec{x}, 2y + 1) = H_2(\vec{f}, \vec{x}, y, F(\vec{f}, \vec{x}, y)), \quad (13)$$

$$|F(\vec{f}, \vec{x}, y)| \leq |K(\vec{f}, \vec{x}, y)| \quad (14)$$

The last condition is equivalent to the condition  $F(\vec{f}, \vec{x}, y) \leq K^*(\vec{f}, \vec{x}, y)$  for  $K^* = (1 \# K(\vec{f}, \vec{x}, y)) \div 1$ .

Townsend ([13]) considered the least class of functionals which contains the polynomial time computable functions, the application functional  $Ap$  defined by  $Ap(f, x) = f(x)$ , which is closed under expansion, functional composition, functional substitution and limited recursion on notation. He has also shown that the scheme of functional substitution is redundant; thus we introduce the following appropriately modified definition.

**Definition 3** *The class  $\mathcal{B}$  of basic feasible functionals is the least class of functionals which contains initial functions  $\mathbf{o}(x) = 0$ ,  $s_0(x) = 2x$ ,  $s_1(x) = 2x + 1$ ,  $i_k^n(x_1, \dots, x_n) = x_k$ ,  $x \# y$  and the application functional  $Ap(f, x) = f(x)$ , which is closed under expansion, functional composition and limited recursion on notation.*

The class of basic feasible functionals extends the class of feasible functions in a “minimal way”; it is obtained using essentially the same closure conditions as the class of feasible functions (taking into account that the schema of functional substitution is redundant and so it can be omitted). The only difference is addition of the application functional to the set of basic functions. However, in any reasonable model of feasibility in higher types such functional must be considered feasible. Also, it is easy to see that the only functions which belong to the class of the basic feasible functionals are in fact just feasible functions.

As before, recursion on notation of type (12) together with (13) can be replaced with a single condition

$$F(\vec{f}, \vec{x}, y) = H(\vec{f}, \vec{x}, y, F(\vec{f}, \vec{x}, \lfloor \frac{1}{2}y \rfloor)).$$

However, the growth rate of a basic feasible functional clearly cannot be majorized by a first order polynomial. Thus, we cannot expect to have simple analogues for schemas involving polynomial bounds or limited primitive recursion. The same applies to the machine models of feasibility in higher types. This is why we need the following definitions introduced by Kapron in [5].

**Definition 4** *The functional of type  $\mathbb{N}^{\mathbb{N}} \rightarrow \mathbb{N}^{\mathbb{N}}$  defined such that  $f \mapsto |f|$  where*

$$|f|(x) = \max_{|y| \leq x} |f(y)|$$

*is called the norm functional; the function  $|f|$  is called the norm of the function  $f$ .*

Kapron has shown that the functional of type (1,1) such that  $\langle f, x \rangle \mapsto |f|(|x|)$  is not basic feasible and that, in fact, it cannot be majorized by any basic feasible functional.

The class of *second-order polynomials* was also introduced in [5]. Second order polynomials play quite the same role which first order polynomials play for feasible functions of type  $\mathbb{N}^k \rightarrow \mathbb{N}$ . However, there is a major difference in their nature which greatly complicates their applications: while first order polynomials are themselves feasible functions, the above result of Kapron implies that the second order polynomials are *not* feasible functionals.

**Definition 5** *Let  $x_0, x_1, \dots$  and  $f_0, f_1, \dots$  be sets of first and second order variables respectively; then the set of second order polynomials in  $|f_0|, |f_1|, \dots$  and  $|x_0|, |x_1|, \dots$  is defined inductively as the least set of terms of the language  $L_P$  containing constants  $\underline{n}$  for each natural number  $n$  and all terms  $|x_0|, |x_1|, \dots$  and which satisfies the following closure condition: if  $P, Q$  are second-order polynomials and  $f_i$  is a second-order variable, then  $P + Q$ ,  $P \cdot Q$  and  $|f_i|(P)$  are also second-order polynomials.*

Clearly, second order polynomials take into account the growth rate of functions. However, they cannot be considered to be feasible: to compute  $|f|(|x|)$ , we must compute  $|f(y)|$  for all  $y$  such that  $|y| \leq |x|$ , which is exponentially many values of  $y$  (in  $|x|$ ). Kapron's result showing that the norm is not basic feasible implies that the same holds of second order polynomials.

We can now state the main results of this paper, which are the best possible analogues of the corresponding first order theorems. Due to the mentioned difficulty (non-feasibility of the second order polynomials), the proofs of these analogues use formal logic as an essential tool.

**Definition 6** Let  $Q(|f|, |\vec{x}|)$  be a second order polynomial and let  $G(f, \vec{x})$  and  $H(f, \vec{x}, z, y)$  be two basic feasible functionals. Assume that the functional  $F(f, \vec{x}, y)$  satisfies

$$\begin{aligned} F(f, \vec{x}, 0) &= G(f, \vec{x}) \\ F(f, \vec{x}, y) &= H(f, \vec{x}, F(f, \vec{x}, \lfloor \frac{1}{2}y \rfloor), y), \\ |F(f, \vec{x}, y)| &\leq Q(|f|, |\vec{x}|, |y|) \end{aligned} \tag{15}$$

Then we say that  $F$  is defined from functionals  $G, H$  by *polynomially bounded recursion on notation with the bound  $Q$* .

**Theorem:** Assume that the functional  $F(f, \vec{x}, y)$  is defined from the functionals  $G(f, \vec{x})$  and  $H(f, \vec{x}, z, y)$  by *polynomially bounded recursion on notation with the bound  $Q(|f|, |\vec{x}|, |y|)$* . Then  $F(f, \vec{x}, y)$  is a basic feasible functional.

**Definition 7** Let  $P(|f|, |\vec{x}|)$  and  $Q(|f|, |\vec{x}|)$  be two second order polynomials and let  $G(f, \vec{x})$  and  $H(f, \vec{x}, y, z)$  be two basic feasible functionals. Assume that the functional  $F^*(f, \vec{x}, y)$  satisfies

$$\begin{aligned} F^*(f, \vec{x}, 0) &= G(f, \vec{x}); \\ F^*(f, \vec{x}, y + 1) &= H(f, \vec{x}, F^*(f, \vec{x}, y), y); \\ (\forall y \leq P(|f|, |\vec{x}|)) &(|F^*(f, \vec{x}, y)| \leq Q(|f|, |\vec{x}|)); \\ (\forall y)(y \geq P(|f|, |\vec{x}|) \rightarrow &F^*(f, \vec{x}, y) = F^*(f, \vec{x}, P(|f|, |\vec{x}|))); \\ (\forall f)(\forall f^*)(\forall y \leq P(|f|, |\vec{x}|)) &(F(f, \vec{x}, y) = F(f^*, \vec{x}, y)) \rightarrow \\ &(\forall y)(F(f, \vec{x}, y) = F(f^*, \vec{x}, y)); \end{aligned}$$

and that the functional  $F(f, \vec{x})$  is defined by

$$F(f, \vec{x}) = F^*(f, \vec{x}, P(|f|, |\vec{x}|)). \tag{16}$$

Then we say that  $F(f, \vec{x})$  is defined from functionals  $G, H$  by *polynomially bounded recursion of polynomial length with bounds  $(Q, P)$* .

**Theorem:** Assume that the functional  $F(f, \vec{x})$  is defined from the functionals  $G$  and  $H$  by *polynomially bounded recursion of polynomial length with bounds  $(Q, P)$* . Then the functional  $F(f, \vec{x})$  is a basic feasible functional.

## 1.2 Turing Machine Characterization

We use the usual model for computability with oracle Turing machines (OTM). Function inputs are presented using oracles, which are queried using separate *write-only oracle input tapes* and *read-only oracle output tapes*, while the machine is in the *oracle query state* corresponding to the input function which is

queried. To query function input  $f$  at the value  $x$ ,  $x$  is written in binary notation on the oracle input tape associated with  $f$ , and the corresponding oracle query state is entered. After entering the oracle state which corresponds to  $f$ , the value  $f(x)$  appears on the oracle output tape associated with  $f$ , the oracle input tape is then erased and both the writing head of the oracle input tape and the reading head of the oracle output tape are placed at the corresponding initial cells of the tapes. Thus, iterations of the form  $f(f(\dots f(x)\dots))$  cannot be computed without the machine having to copy the intermediate results from the oracle output tape to the oracle input tape. In general, there are two possible conventions for accounting for the running time of an oracle call. In Mehlhorn's model, an oracle call has unit cost, while in the Kapron/Cook model, the oracle call described above has a cost of  $|f(x)|$  time steps. Mehlhorn [9] and Cook and Kapron [5] proved the following theorems.

**Theorem** (Mehlhorn, [9]): A functional  $F(\vec{f}, \vec{x})$ , where  $\vec{f} \in (\mathbb{N}^{\mathbb{N}})^k$  and  $\vec{x} \in \mathbb{N}^l$  is basic feasible if and only if there exists an oracle Turing machine  $M$  with oracles for functions  $\vec{f}$  and a basic feasible functional  $K(\vec{f}, \vec{x})$  such that  $M$  computes  $F(\vec{f}, \vec{x})$  and the running time  $T(\vec{f}, \vec{x})$ , with a unit cost for each oracle query, satisfies

$$(\forall \vec{f})(\forall \vec{x})(T(\vec{f}, \vec{x}) \leq K(\vec{f}, \vec{x}))$$

**Theorem** (Kapron and Cook, [5]): A functional  $F(\vec{f}, \vec{x})$ , where  $\vec{f} \in (\mathbb{N}^{\mathbb{N}})^k$  and  $\vec{x} \in \mathbb{N}^l$  is basic feasible if and only if there exists an oracle Turing machine  $M$  with oracles for functions  $\vec{f}$  and a second order polynomial  $P(|\vec{f}|, |\vec{x}|)$  such that  $M$  computes  $F(\vec{f}, \vec{x})$  and the running time  $T(\vec{f}, \vec{x})$ , with  $|f_i(z)|$  as the cost for an oracle query of  $f_i \in \vec{f}$  at oracle input value  $z$ , satisfies

$$(\forall \vec{f})(\forall \vec{x})(T(\vec{f}, \vec{x}) \leq P(|\vec{f}|, |\vec{x}|))$$

In this paper we improve both theorems by combining their best features.

**Definition 8** A functional  $F(\vec{f}, \vec{x})$  is computable in polynomial time if there exists an oracle Turing machine  $M$  with oracles for functions  $\vec{f}$  and a second order polynomial  $P(|\vec{f}|, |\vec{x}|)$  such that  $M$  computes  $F(\vec{f}, \vec{x})$  and for all  $\vec{f}, \vec{x}$ , the running time  $T(\vec{f}, \vec{x})$ , obtained by counting each oracle query as a single step regardless of the size of the oracle output, satisfies

$$T(\vec{f}, \vec{x}) \leq P(|\vec{f}|, |\vec{x}|).$$

**Theorem(10):** A functional  $F(\vec{f}, \vec{x})$  is a polynomial time computable functional if and only if it is a basic feasible functional.

### 1.3 Formal Theories of Second Order Arithmetic

Theories  $\mathbf{S}_2^1$  and  $\mathbf{R}_2^1$  which we use in this paper were introduced in [6] and used in [3].  $\mathbf{S}_2^1$  is a second order extension of Buss's  $S_2^1$ . Theory  $\mathbf{S}_2^1$  is formulated in the language of  $S_2^1$ , i.e.  $\leq, 0, 1, +, \cdot, |x|, \lfloor \frac{1}{2}x \rfloor, \#$ , extended (for easier bootstrapping) by the function  $x \upharpoonright y$  producing the number consisting of the first, more significant  $y$  bits of  $x$  i. e.  $x \upharpoonright y = \lfloor x/2^{|x|-y} \rfloor$ , and by a symbol for the application functional  $Ap(f, \vec{x})$ . Consequently, we also extend the usual set of open axioms *BASIC* of theories of bounded arithmetic by adding a few axioms for  $\upharpoonright$  (we keep the same notation for the extended set).

The hierarchy of bounded formulas (i.e. formulas whose all quantifiers are bounded) is obtained from the corresponding hierarchies of bounded formulas of the first order bounded arithmetic, by allowing the application functional  $Ap(f, \vec{x})$  to appear in the atomic formulas. Bounded quantifiers of the form  $\exists x \leq |t|$  and  $\forall x \leq |t|$  are called *sharply bounded* quantifiers. Bounded formulas form a hierarchy at whose bottom level are *sharply bounded* or  $\Sigma_0^b$  formulas, i.e. formulas whose all quantifiers are sharply bounded. The complexity of an arbitrary bounded formula (e.g.,  $\Sigma_i^b$ ,  $\Pi_i^b$ ) is obtained by counting the alternations of bounded quantifiers ignoring sharply bounded ones.

Theory  $\mathbf{S}_2^i$  is obtained from *BASIC* by adding either one of the following two induction schemas for  $\Sigma_i^b$  formulas:

$$\Sigma_i^b\text{-PIND} \quad A(0, \vec{f}, \vec{y}) \wedge (\forall x)(A(\lfloor \frac{1}{2}x \rfloor, \vec{f}, \vec{y}) \rightarrow A(x, \vec{f}, \vec{y})) \rightarrow (\forall x)A(x, \vec{f}, \vec{y})$$

$$\Sigma_i^b\text{-LIND} \quad A(0, \vec{f}, \vec{y}) \wedge (\forall x)(A(x, \vec{f}, \vec{y}) \rightarrow A(x+1, \vec{f}, \vec{y})) \rightarrow (\forall x)A(|x|, \vec{f}, \vec{y})$$

The proof of the fact that theories  $(\text{BASIC} + \Sigma_1^b\text{-PIND})$  and  $(\text{BASIC} + \Sigma_1^b\text{-LIND})$  are equivalent is not only standard, (see [1]); but is actually facilitated by the presence of the function  $x \upharpoonright y$  in our language and its basic properties as axioms of *BASIC*. We will also mention another sequence of formal theories,  $\mathbf{R}_2^i$ , important for the study of parallel computability of functionals, obtained from (again slightly extended) theory *BASIC* by adding the following induction schema for  $\Sigma_i^b$  formulas:

$$\Sigma_i^b\text{-LPIND} \quad A(0, \vec{f}, \vec{y}) \wedge (\forall x)(A(\lfloor \frac{1}{2}x \rfloor, \vec{f}, \vec{y}) \rightarrow A(x, \vec{f}, \vec{y})) \rightarrow (\forall x)A(|x|, \vec{f}, \vec{y})$$

First order theory  $R_2^1$  was studied by Allen ([8]) and, independently, by Clote and Takeuti ([2]) (*TNC* in their notation). Our techniques apply equally well to parallel computable functionals (a generalization of the class *NC*; see [3]).

Since we do not have functional substitution in our definition of basic feasible functionals, theories  $\mathbf{S}_2^i$  have no comprehension axioms at all. The most important property of the above theories is that definable function(al)s of these theories correspond to important complexity classes of functionals.

**Definition 9** A functional  $F$  is  $\Sigma_i^b$  definable in the theory  $\mathbf{S}_2^i$ ,  $i \geq 1$ , if there exists a term  $t(\vec{f}, \vec{x})$  and a  $\Sigma_i^b$  formula  $\Psi_F(\vec{f}, \vec{x}, y)$  such that

$$\mathbf{S}_2^i \vdash \forall \vec{f}, \vec{x} \exists! y \leq t(\vec{f}, \vec{x}) \Psi_F(\vec{f}, \vec{x}, y)$$

and

$$\langle \mathbb{N}^k, (\mathbb{N}^{\mathbb{N}})^m \rangle \models \forall \vec{f}, \vec{x} \Psi_F(\vec{f}, \vec{x}, F(\vec{f}, \vec{x})).$$

For  $i \geq 1$  provably  $\Sigma_i^b$  definable functionals of  $\mathbf{S}_2^i$  are functionals of the second order polynomial time hierarchy,  $\Pi_i^P$ , a natural extension of the usual polynomial time hierarchy for functions  $\Pi_i^P$ . In particular,  $\Sigma_1^b$  definable functionals of the theory  $\mathbf{S}_2^1$  are exactly functionals which form the first level of this hierarchy,  $\Pi_1^P$  which are just the basic feasible functionals. In order to prove this fact we use *multiple limited recursion on notation* (MLRN) due to Cook and Kapron ([5]), of which we give a very simple proof. Our argument for  $i = 1$  easily generalizes to  $\mathbf{S}_2^i$  and  $\Pi_i^P$  for any  $i$ .

## 1.4 Proof Techniques

The main novelty of this paper is in the way how we use logic (in particular formal theories) as a tool to derive results of recursion-theoretic nature as well as results on Turing computability of functionals.

It is easy to see that every basic feasible functional  $F(x, f)$  is computable on a polynomial time oracle Turing machine, as well as that it can be obtained using polynomially bounded recursion of polynomial length; this follows immediately from the fact that every basic feasible functional can be majorized by a second order polynomial. The difficult part is to prove that if a functional is polynomial time computable or computable from basic feasible functionals using polynomially bounded recursion of polynomial length, that then it is a basic feasible functional.

The approach used in the original Cobham's proof for the first order case is not applicable here, since it is not possible to code directly a complete instantaneous description of an oracle Turing machine run or of the sequence of intermediate values of a recursive procedure after  $|i|$  many steps and then evaluate this functional at a value of  $|i|$  which is greater or equal than the number of necessary steps for the computation to terminate. This is because both the sizes of complete instantaneous descriptions of an oracle Turing machine run (or of the sequence of intermediate values of a recursive procedure) and the number of steps necessary for termination of a computation of an oracle Turing machine (or of a recursive procedure) are bounded by second order polynomials which cannot be majorized by basic feasible functionals. However, definitions by recursion on notation require such bounds.

We eliminate the need for such bounds by showing that functionals computable on a Turing machine or by a recursive procedure with bounds which are second order polynomials are  $\Sigma_1^b$  definable in the theory  $\mathbf{S}_2^1$ , and then, using entirely conventional proof-theoretic methods, we show that functionals which are  $\Sigma_1^b$  definable in the theory  $\mathbf{S}_2^1$  are exactly basic feasible functionals.

In order to show that functionals defined using second order polynomials are  $\Sigma_1^b$  definable in the theory  $\mathbf{S}_2^1$  we replace bounds involving second order



polynomials with their suitable representations within our formal theory  $\mathbf{S}_2^1$ . This representation consists of a sequence of existentially bounded quantifiers prefixing a term bound, and is based on the following Lemma. Without any loss of generality, from now on we assume that we have only one input function  $f$ .

**Lemma:** *Let  $P(|f|, |\vec{x}|)$  be a second order polynomial of depth  $d$  (see section 2); then there exists a sequence of terms  $t_0, \dots, t_d$  containing only operations  $Ap(f, x), x + y, x \cdot y, x \# y, \lfloor \frac{1}{2}x \rfloor$  and the constant  $\underline{1}$  such that for all  $f$  and all  $\vec{x}$  the following four formulas are true in the standard model:*

$$(\forall u)(|u| \leq P(|f|, |\vec{x}|) \leftrightarrow (\exists z_1 \leq t_0(\vec{x})) \dots (\exists z_d \leq t_d(f, \vec{x}, z_1, \dots, z_{d-1})) \\ (u \leq t_d(f, \vec{x}, z_1, \dots, z_d)))$$

Thus, a bound involving a second order polynomial can be replaced by a sequence of existential quantifiers bounded by terms.

We now work in a formal theory and can use objects defined by formulas with higher quantifier complexity as well. Also, instead of having to provide explicit definitions with appropriate bounds, we can use induction and give “existential” proofs. This greatly facilitates our arguments.

## 2 Second order polynomials

The *depth*  $d(P)$  of a second order polynomial is defined (in [5]) to be the maximal number of nestings of the application functional; thus, the depth of the polynomials not involving the application functional is equal to 0 and  $d(P_1 + P_2) = d(P_1 \cdot P_2) = \max\{d(P_1), d(P_2)\}$ , while  $d(f(P_1)) = d(P_1) + 1$ .

The following simple Lemma can be proved by induction on the complexity of the definition of  $\mathcal{B}$  functionals, using monotonicity of the function  $\lambda z. |f|(z)$ .

**Lemma 1** (Townsend [13]) *For every functional  $F(\vec{f}, \vec{x}) \in \mathcal{B}$  there exists a second order polynomial  $P(|\vec{f}|, |\vec{x}|)$  such that for all  $\vec{f}$  and for all  $\vec{x}$*

$$|F(\vec{f}, \vec{x})| \leq P(|\vec{f}|, |\vec{x}|).$$

In order to handle bounds with second order polynomials using proof-theoretic means, we inessentially narrow the class of second order polynomials which we will be using. We consider only polynomials which have the property that for every  $m$  smaller than the depth of the polynomial  $P$  and every  $f_j$  appearing in  $P$ , there exists exactly one sub-polynomial  $P^*$  of  $P$  of the form  $f_j(P^*)$  of depth  $m$ ; also, for any two subpolynomials of depths  $d_1$  and  $d_2$  if  $d_1 \leq d_2$  then for all  $\vec{f}$  and all  $\vec{x}$ ,  $P_{d_1}(|\vec{f}|, |\vec{x}|) \leq P_{d_2}(|\vec{f}|, |\vec{x}|)$ . Such polynomials are called *regular* polynomials. Every second order polynomial is majorized by a regular second order polynomial. To see this just observe that for every function  $f$ , the function  $\lambda z. |f|(z)$  is a monotone increasing function in  $z$  and so we can

inductively replace several second order polynomials which are arguments of functions  $|f_j|$  and which are of the same depth, by their sum. Since we will use second order polynomials only to majorize other functionals, we will work only with regular second order polynomials. From now on, when we say that  $P$  is a second order polynomial, we actually mean that  $P$  is a *regular* second order polynomial. Also, only for the simplicity of our notation, we will assume that we have only one second order variable  $f$ . It is easy to see that all our arguments easily generalize to the cases involving several second order variables.

The next lemma will be used to replace bounds which are second order polynomials with second order terms involving only feasible functionals. We use “witnessing points” to approximate from below the value of a second order polynomial.

**Lemma 2** *Let  $P(|f|, |\vec{x}|)$  be a second order polynomial of depth  $d$ ; then there exists a sequence of terms  $t_0, \dots, t_d$  containing only operations  $Ap(f, x), x+y, x \cdot y, x \# y, \lfloor \frac{1}{2}x \rfloor$  and the constant  $\underline{1}$  such that for all  $f$ , all  $\vec{x}$  and all  $u$ :*

$$|u| \leq P(|f|, |\vec{x}|) \leftrightarrow (\exists z_1 \leq t_0(\vec{x}))(\exists z_2 \leq t_1(f, \vec{x}, z_1)) \dots \\ (\exists z_d \leq t_{d-1}(f, \vec{x}, z_1, \dots, z_{d-1}))(u \leq t_d(f, \vec{x}, z_1, \dots, z_d))$$

**Proof:** Simple.

**Definition 10** *Let  $P(|f|, |\vec{x}|)$  be a second order polynomial and  $t_0(\vec{x}), \dots, t_d(f, \vec{x}, z_1, \dots, z_d)$  be the sequence of terms as in Lemma (2); then we call this sequence the sequence of terms associated with the polynomial  $P(|f|, |\vec{x}|)$ . We will often abbreviate the sequence of these terms as  $t^{\vec{P}}$ , and the corresponding quantifier prefix  $(\exists z_1 \leq t_0(\vec{x}))(\exists z_2 \leq t_1(f, \vec{x}, z_1)) \dots (\exists z_d \leq t_{d-1}(f, \vec{x}, z_1, \dots, z_{d-1}))$  as  $(\exists \vec{z} \leq t^{\vec{P}})$ .*

### 3 Theories of Second Order Arithmetic

We now define two second order formal theories of arithmetic  $\mathbf{S}_2^1$  and  $\mathbf{R}_2^1$ ; one sort of variables range over the set of natural numbers  $\mathbb{N}$ ; the other range over the set of functions of type  $\mathbb{N} \rightarrow \mathbb{N}$ . We put no constraints on the growth rate of functions. These two theories are then used to characterize type 2 feasible functionals in the same way how Buss’s  $S_2^1$  is used to characterize feasible functions.

We will use Buss’s results on introducing the polynomial time computable functions in  $S_2^1$  (see [1]), but due to the presence of functions which can be of an arbitrary growth rate, we must do more work. Also, to facilitate the bootstrapping of our theories, we will add a few more polynomial time functions to the language of our theories which are second order versions of Buss’s  $S_2^1$ .

**Definition 11** We will denote by  $L_b^2$  the language consisting of the following symbols  $\leq, 0, 1, +, \cdot, |x|, \lfloor \frac{1}{2}x \rfloor, \#, \upharpoonright$  and  $Ap(f, \vec{x})$ . Here  $x \upharpoonright y$  denotes<sup>2</sup> the  $y$  most significant bits of  $x$ .  $Ap(f, \vec{x})$  is the only mixed-sort symbol for the application functional whose value is  $f(\vec{x})$  in the standard interpretation.

We will consider the following induction schemas.

$$\Sigma_1^b\text{-PIND} \quad A(0, \vec{\alpha}) \wedge (\forall x)(A(\lfloor \frac{1}{2}x \rfloor, \vec{\alpha}) \rightarrow A(x)) \rightarrow (\forall x)A(x, \vec{\alpha})$$

$$\Sigma_1^b\text{-LIND} \quad A(0, \vec{\alpha}) \wedge (\forall x)(A(x, \vec{\alpha}) \rightarrow A(x+1, \vec{\alpha})) \rightarrow (\forall x)A(|x|, \vec{\alpha})$$

$$\Sigma_i^b - LPIND \quad A(0, \vec{\alpha}) \wedge (\forall x)(A(\lfloor \frac{1}{2}x \rfloor, \vec{\alpha}) \rightarrow A(x, \vec{\alpha})) \rightarrow (\forall x)A(|x|, \vec{\alpha})$$

$$\Sigma_i^b - LLIND \quad A(0, \vec{\alpha}) \wedge (\forall x)(A(x, \vec{\alpha}) \rightarrow A(x+1, \vec{\alpha})) \rightarrow (\forall x)A(\|x\|, \vec{\alpha})$$

One can easily check that by adding to *BASIC* the following axioms for  $x \upharpoonright y$ :  $x \upharpoonright 0 = 0, x \geq 1 \rightarrow x \upharpoonright 1 = 1, y < |x| \rightarrow x \upharpoonright y = \lfloor \frac{x \upharpoonright (y+1)}{2} \rfloor$  and  $y \geq |x| \rightarrow x \upharpoonright y = x$  makes the usual proof (see Buss's [1]) of *BASIC* +  $(\Sigma_i^b\text{-LIND}) \equiv \text{BASIC} + (\Sigma_i^b\text{-PIND})$  simple, and that it goes through also for  $i = 1$  equally easily. Theories  $\mathbf{S}_2^i$  are obtained by adding to such slightly extended *BASIC* either  $(\Sigma_i^b\text{-LIND})$  or  $(\Sigma_i^b\text{-PIND})$ . Theories  $\mathbf{R}_2^i$  are obtained by first extending the language  $L_b^2$  into the language  $L_d^2 = L_b^2 \cup \{-, Bit(x, y), Lsp(x, y)\}$ ; then by expanding the axioms of *BASIC* into *BASIC*<sup>+</sup> as described in [8], by adding basic properties of these functions and finally adding either of the two schemas  $\Sigma_i^b - LPIND$  or  $\Sigma_i^b - LLIND$ .

We now concentrate on theory  $(\Sigma_1^b - PIND)$  as our main tool for studying the properties of basic feasible functionals. Let  $\vec{\alpha} = \vec{f}, \vec{x}$ . We now want to show that a functional is a basic feasible functional if and only if it is  $\Sigma_1^b$  definable in the theory  $(\Sigma_1^b - PIND)$ . We start with the easier direction.

**Theorem 3** Every basic feasible functional is  $\Sigma_1^b$  definable in the theory  $(\Sigma_1^b - PIND)$ .

**Proof:** We prove by induction on complexity of definition of a functional  $F$  that there exists a  $\Sigma_1^b$  formula  $\Theta_F(\vec{\alpha}, \vec{z}_1, \dots, \vec{z}_k, y)$  and sequences of terms  $\vec{t}_1, \dots, \vec{t}_k, t_{k+1}$  such that

$$\begin{aligned} (\Sigma_1^b - PIND) \vdash \forall \vec{\alpha} \exists \vec{z}_1 \leq \vec{t}_1(\vec{\alpha}) \dots \exists \vec{z}_k \leq \vec{t}_k(\vec{\alpha}, \vec{z}_1, \dots, \vec{z}_{k-1}) \\ \exists! y \leq t_{k+1}(\vec{\alpha}, \vec{z}_1, \dots, \vec{z}_k) \Theta_F(\vec{\alpha}, \vec{z}_1, \dots, \vec{z}_k, y) \end{aligned} \quad (17)$$

which clearly implies that  $F$  is  $\Sigma_1^b$  definable in  $(\Sigma_1^b - PIND)$ . The proof proceeds by induction on the definition of  $F \in \mathcal{B}$ . If  $F$  is obtained by functional composition or by expansion the proof is straightforward.

If  $F$  is defined by limited recursion on notation from  $G, H$  and  $K$ , we use Buss's function  $SqBd(a, b) = (2b+1)\#(4(2a+1)^2)$  (see [1]), which puts an upper

<sup>2</sup>Notice that  $x \upharpoonright y$  is  $MSP(x, y)$  in Buss's notation which we here conveniently simplify.

bound on codes of sequences of length at most  $|b| + 1$ , consisting of numbers  $\leq a$ . Then, assuming that

$$\begin{aligned} F(\vec{\alpha}, 0) &= G(\vec{\alpha}) \\ F(\vec{\alpha}, y) &= H(\vec{\alpha}, y, F(\vec{\alpha}, \lfloor \frac{1}{2}y \rfloor)) \\ F(\vec{\alpha}, y) &\leq K(\vec{\alpha}, y) \end{aligned}$$

by induction hypothesis there are formulas  $\Theta_H, \Theta_G, \Theta_K$  and terms  $\vec{t}_i^H, \vec{t}_j^G, \vec{t}_r^K$ ,  $1 \leq i, j, r \leq k+1$  such that

$$\begin{aligned} (\Sigma_1^b - PIND) &\vdash \forall \vec{\alpha} (\exists \vec{z}^G \leq \vec{t}^G) \exists g \leq t_{k+1}^G(\vec{\alpha}, \vec{z}^G) \Theta_G(\vec{\alpha}, \vec{z}^G, g), \\ (\Sigma_1^b - PIND) &\vdash \forall \vec{\alpha} \forall y \forall v (\exists \vec{z}^H \leq \vec{t}^H) \exists h \leq t_{k+1}^H(\vec{\alpha}, y, v, \vec{z}^H) \Theta_H(\vec{\alpha}, y, v, \vec{z}^H, h) \\ (\Sigma_1^b - PIND) &\vdash \forall \vec{\alpha} \forall y (\exists \vec{z}^K \leq \vec{t}^K) \exists q \leq t_{k+1}^K(\vec{\alpha}, y, \vec{z}^K) \Theta_K(\vec{\alpha}, y, \vec{z}^K, q). \end{aligned} \tag{18}$$

then we prove

$$\begin{aligned} (\Sigma_1^b - PIND) &\vdash \forall \vec{\alpha} \forall y (\exists \vec{z}^K \leq \vec{t}^K) \exists t \leq y \exists w \leq SqBd(t_{k+1}^K(\vec{\alpha}, t, \vec{z}^K), y) \\ &\quad \exists z \leq w \Theta_F(\vec{\alpha}, y, t, w, \vec{z}^K, z) \end{aligned}$$

where  $\Theta_F$  is the conjunction of the following formulas:

$$\begin{aligned} &(\exists \vec{z}^G \leq \vec{t}^G) \exists g \leq t_{k+1}^G(\vec{\alpha}, \vec{z}^G) (\Theta_G(\vec{\alpha}, \vec{z}^G, g) \wedge \exists q_0 \leq t_{k+1}^K(\vec{\alpha}, t, \vec{z}^K) \\ &\quad \Theta_K(\vec{\alpha}, 0, \vec{z}^K, q_0) \wedge (w)_0 = \min(g, q_0)) \end{aligned}$$

$$\begin{aligned} &(\forall i < |y| \exists q_{i+1} \leq t_{k+1}^K(\vec{\alpha}, t, \vec{z}^K) \Theta_K(\vec{\alpha}, y \upharpoonright (i+1), \vec{z}^K, q_{i+1})) \\ &\wedge (w)_i \leq t_{k+1}^K(\vec{\alpha}, t, \vec{z}^K) \wedge (\exists \vec{z}^H \leq \vec{t}^H) \exists h \leq t_{k+1}^H(\vec{\alpha}, y \upharpoonright i, (w)_i, \vec{z}^H) \\ &\quad \Theta_H(\vec{\alpha}, y \upharpoonright i, (w)_i, \vec{z}^H, h) \wedge (w)_{i+1} = \min(h, q_{i+1})) \wedge ((w)_{|y|} = z) \end{aligned}$$

$$(\forall j \leq |y| \exists q \leq t_{k+1}^K(\vec{\alpha}, t, \vec{z}^K) \Theta_K(\vec{\alpha}, y \upharpoonright j, \vec{z}^K, q))$$

The proof is straightforward; the first formula corresponds the initial value of the function the second to the recursion on notation and the third to bounding and the final value of the computation.

To prove the converse we need definitions by *multiple limited recursion on notation* MLRN due to Cook and Kapron, of which we give a very simple proof.

**Theorem 4** *Let  $G_1, G_2, H_1, H_2, K_1$  and  $K_2$  be basic feasible functionals, and assume that  $F_1$  and  $F_2$  satisfy*

$$F_1(0, \vec{\alpha}) = G_1(\vec{\alpha})$$

$$\begin{aligned}
F_2(0, \vec{\alpha}) &= G_2(\vec{\alpha}) \\
F_1(u, \vec{\alpha}) &= H_1(u, F_1(\lfloor \frac{1}{2}u \rfloor, \vec{\alpha}), F_2(\lfloor \frac{1}{2}u \rfloor, \vec{\alpha}), \vec{\alpha}) \\
F_2(u, \vec{\alpha}) &= H_2(u, F_1(\lfloor \frac{1}{2}u \rfloor, \vec{\alpha}), F_2(\lfloor \frac{1}{2}u \rfloor, \vec{\alpha}), \vec{\alpha}) \\
F_1(u, \vec{\alpha}) &\leq K_1(u, \vec{\alpha}) \\
F_2(u, \vec{\alpha}) &\leq K_2(u, \vec{\alpha}, F_1(u, \vec{\alpha}))
\end{aligned} \tag{19}$$

Then  $F_1$  and  $F_2$  are also basic feasible functionals.

**Proof:** Let  $\hat{K}$  be defined as follows:

$$\begin{aligned}
\hat{K}(0, \vec{\alpha}) &= 0 \\
\hat{K}(u, \vec{\alpha}) &= \begin{cases} \hat{K}(\lfloor \frac{1}{2}u \rfloor, \vec{\alpha}) & \text{if } K_1(u, \vec{\alpha}) \leq K_1(\hat{K}(\lfloor \frac{1}{2}u \rfloor, \vec{\alpha}), \vec{\alpha}) \\ u & \text{otherwise} \end{cases} \\
\hat{K}(u, \vec{\alpha}) &\leq u
\end{aligned}$$

and let  $\bar{K}(u, \vec{\alpha}) = K_1(\hat{K}(u, \vec{\alpha}), \vec{\alpha})$ . Thus, as it is easily proved by induction on  $u$ ,  $\bar{K}(u, \vec{\alpha})$  is the largest of the value of  $K_1(u \upharpoonright i, \vec{\alpha})$ , for  $u$  fixed and  $0 \leq i \leq |u|$ . Thus, the function

$$S(u, \vec{\alpha}) = SqBd(\bar{K}(u, \vec{\alpha}), u)$$

has the property that it bounds any sequence  $a$  of length  $|u|$  satisfying:  $(\forall i \leq |u|)((a)_i \leq K_1(u \upharpoonright i, \vec{\alpha}))$ .

Consider now functionals  $F$  and  $W$  defined as follows:

$$\begin{aligned}
F(0, \vec{\alpha}, w) &= G_2(\vec{\alpha}) \\
F(u, \vec{\alpha}, w) &= \min\{H_2(u, (w)_{(|u|-1)}), F(\lfloor \frac{1}{2}u \rfloor, \vec{\alpha}, w), K_2(u, \vec{\alpha}, (w)_{|u|})\}
\end{aligned}$$

and

$$\begin{aligned}
W(0, \vec{\alpha}) &= \langle G_1(\vec{\alpha}) \rangle \\
W(u, \vec{\alpha}) &= \min\{W(\lfloor \frac{1}{2}u \rfloor, \vec{\alpha}) \cap H_1(u, (W(\lfloor \frac{1}{2}u \rfloor, \vec{\alpha}))_{\lfloor \frac{1}{2}|u| \rfloor}), \\
&\quad F(\lfloor \frac{1}{2}u \rfloor, \vec{\alpha}, W(\lfloor \frac{1}{2}u \rfloor, \vec{\alpha}), \vec{\alpha}), S(u, \vec{\alpha})\}
\end{aligned} \tag{20}$$

where  $w \frown a$  stands for the sequence  $w$  extended at the end with an extra term. Then, clearly,

$$F(u, \vec{\alpha}, w) \leq K_2(u, \vec{\alpha}, (w)_{|u|}) \tag{21}$$

$$W(u, \vec{\alpha}) \leq S(u, \vec{\alpha}) \tag{22}$$

which implies that these functionals are both basic feasible. Let now

$$F_1(u, \vec{\alpha}) = (W(u, \vec{\alpha}))_{|u|} \tag{23}$$

$$F_2(u, \vec{\alpha}) = F(u, \vec{\alpha}, W(u, \vec{\alpha})) \tag{24}$$

then one can easily prove by (polynomial) induction on  $u$  that these functionals satisfy the recursive schema (19). The theorem easily generalizes to  $n > 2$  functionals, in which case we proceed by induction and assume that the claim of the Lemma holds for  $n - 1$ . Replace the first functional,  $F_1$ , by a variable  $w$ , representing the code of the computation of a functional  $W$ , as we did in the case of  $n = 2$ ; hence the number of the remaining functionals is  $n - 1$ . ■

We now want to show that every functional of type 2 which is  $\Sigma_1^b$  definable in  $(\Sigma_1^b - PIND)$  is a basic feasible functional. The proof of this fact is identical to the Buss proof of the corresponding result for  $S_2^1$  and the polynomial time computable functions, with one single exception. For convenience, instead of the original Buss's proof, we will use Sieg's method of Herbrand Analyses from [10]. The following Lemma replaces Lemma 1.3.4. from [11]. The rest of the proof is identical to the first order case; the fact that we have a two sorted language has no impact on the rest of the proof.

Exactly as in the first order case, we introduce auxiliary theories needed for the proof.

$(QF(\mathcal{B}) - PIND)$  stands for the theory on a language with a symbol for every functional in  $\mathcal{B}$ . Besides the basic axioms it also has defining recursion equations for every such symbol, and induction schema for every open formula.

$(n - \Sigma_1^b) - PIND$  stands for the theory which extends  $(QF(\mathcal{B}) - PIND)$  with the induction schema for formulas in prenex normal form whose prefix has at most  $n$  bounded existential quantifiers and no sharply bounded universal quantifiers.  $\Sigma_1^b$  formulas without any sharply bounded universal quantifiers are called *strict*  $\Sigma_1^b$  formulas.

**Lemma 5** *Let  $\Delta$  be a set containing only existential formulas (with the existential quantifier bounded or unbounded). Then if  $(n - \Sigma_1^b) - PIND \vdash \Delta$ , then also  $(QF(\mathcal{B}) - PIND) \vdash \Delta$ .*

**Proof:** We proceed by induction on the number of applications of the induction rule applied to  $(n - \Sigma_1^b)$ -formulas which can contain at most  $n$  existential quantifiers (i.e. we *do not* count instances of induction rule applied to open formulas) in the I-normal derivations in  $(n - \Sigma_1^b) - PIND$  of sets of  $(n - \Sigma_1^b)$ -formulas. (Recall that a derivation is I-normal if all cuts are on induction or atomic formulas only.)

Let the claim of the Lemma hold for derivations with  $k$  applications of the induction rule. Assume that  $(n - \Sigma_1^b) - PIND \vdash \Delta$  with an I-normal derivation  $d$  with  $k + 1$  such applications of the induction rule; consider a *top-most* instance of the induction rule applied to a  $(n - \Sigma_1^b)$ -formula  $\psi$  (i.e. an application of the induction rule so that all other applications of the induction rule appearing above it in the derivation  $d$  are on open formulas):

$$\frac{\Gamma(\vec{\alpha}), \psi(0, \vec{\alpha}) \quad \Gamma(\vec{\alpha}), \neg\psi(\lfloor \frac{1}{2}b_i \rfloor, \vec{\alpha}), \psi(b_i, \vec{\alpha})}{\Gamma(\vec{\alpha}), \psi(t(\vec{\alpha}), \vec{\alpha})}.$$

We want to prove that we can reduce this application of  $(n - \Sigma_1^b) - \mathbf{PIND}$  to an application of open induction.

If  $n = 2$ , then  $\psi(b_i, \vec{\alpha})$  is of the form

$$\exists z_1 \leq t_0(b_i, \vec{\alpha}) \exists z_2 \leq t_1(b_i, \vec{\alpha}, z_1) \theta(b_i, z_1, z_2, \vec{\alpha}),$$

where  $b_i$  does not appear in the sequence of variables  $\vec{\alpha}$ , and  $\theta$  is open.

Let  $d_1$  and  $d_2$  be the immediate subderivations; by our assumption, they are derivations in the theory  $(\mathbf{QF}(\mathcal{B}) - \mathbf{PIND})$ . We first replace the derivation  $d_2$  leading to the set  $\Gamma(\vec{\alpha}), \neg\psi(\lfloor \frac{1}{2}b_i \rfloor, \vec{\alpha}), \psi(b_i, \vec{\alpha})$  by a derivation  $d'_2$  of the set  $\Gamma(\vec{\alpha}), a \leq t_0(\lfloor \frac{1}{2}b_i \rfloor, \vec{\alpha}) \wedge c \leq t_1(\lfloor \frac{1}{2}b_i \rfloor, \vec{\alpha}, a) \rightarrow \neg\theta(\lfloor \frac{1}{2}b_i \rfloor, a, c, \vec{\alpha}), \exists z_1 \leq t_0(b_i, \vec{\alpha}) \exists z_2 \leq t_1(b_i, \vec{\alpha}, z_1) \theta(b_i, z_1, z_2, \vec{\alpha})$ , where  $a$  and  $c$  are new variables not previously used in  $d$ . As in the first order case, we can find functionals  $\bar{F}_0, \hat{F}_0, \bar{F}_1$  and  $\hat{F}_1$  and derivations  $d_1^*, d_2^*$  of the sets

$$\Gamma(\vec{\alpha}), \bar{F}_0(\vec{\alpha}) \leq t_0(0, \vec{\alpha}) \wedge \hat{F}_0(\vec{\alpha}) \leq t_1(0, \vec{\alpha}, \bar{F}_0(\vec{\alpha})) \wedge \theta(0, \bar{F}_0(\vec{\alpha}), \hat{F}_0(\vec{\alpha}), \vec{\alpha})$$

and

$$\begin{aligned} & \Gamma(\vec{\alpha}), a \leq t_0(\lfloor \frac{1}{2}b_i \rfloor, \vec{\alpha}) \wedge c \leq t_1(\lfloor \frac{1}{2}b_i \rfloor, \vec{\alpha}, a) \rightarrow \neg\theta(\lfloor \frac{1}{2}b_i \rfloor, a, c, \vec{\alpha}), \\ & \bar{F}_1(b_i, a, c, \vec{\alpha}) \leq t_0(b_i, \vec{\alpha}) \wedge \hat{F}_1(b_i, a, c, \vec{\alpha}) \leq t_1(b_i, \vec{\alpha}, \bar{F}_1(b_i, a, c, \vec{\alpha})) \\ & \wedge \theta(b_i, \bar{F}_1(b_i, a, c, \vec{\alpha}), \hat{F}_1(b_i, a, c, \vec{\alpha}), \vec{\alpha}). \end{aligned}$$

Consider now the formula

$$\begin{aligned} \phi(b_i, x, y, \vec{\alpha}) & \equiv x \leq t_0(b_i, \vec{\alpha}) \wedge \\ & y \leq t_1(b_i, \vec{\alpha}, x) \wedge \theta(b_i, x, y, \vec{\alpha}); \end{aligned}$$

then the above sets are of the form

$$\Gamma(\vec{\alpha}), \phi(0, \bar{F}_0(\vec{\alpha}), \hat{F}_0(\vec{\alpha}), \vec{\alpha})$$

and

$$\Gamma(\vec{\alpha}), \neg\phi(\lfloor \frac{1}{2}b_i \rfloor, a, c, \vec{\alpha}), \phi(b_i, \bar{F}_1(b_i, a, c, \vec{\alpha}), \hat{F}_1(b_i, a, c, \vec{\alpha}), \vec{\alpha})$$

respectively. Thus, in order to be able to apply the induction rule for open formulas, we must find two functionals  $\bar{F}$  and  $\hat{F}$  such that after substituting the free variables  $a$  and  $c$  with the functionals  $\bar{F}(b_i, \vec{\alpha})$  and  $\hat{F}(b_i, \vec{\alpha})$ , respectively, in the following formulas:

$$\phi(0, \bar{F}_0(\vec{\alpha}), \hat{F}_0(\vec{\alpha}), \vec{\alpha}), \neg\phi(\lfloor \frac{1}{2}b_i \rfloor, a, c, \vec{\alpha}), \phi(b_i, \bar{F}_1(b_i, a, c, \vec{\alpha}), \hat{F}_1(b_i, a, c, \vec{\alpha}), \vec{\alpha})$$

they become of the form  $\sigma(0, \vec{\alpha})$ ,  $\sigma(\lfloor \frac{1}{2}b_i \rfloor, \vec{\alpha})$  and  $\sigma(b_i, \vec{\alpha})$  respectively. This suggests the following definitions of  $\bar{F}$  and  $\hat{F}$ :

$$\begin{aligned}\bar{F}(0, \vec{\alpha}) &= \bar{F}_0(\vec{\alpha}) \\ \bar{F}(b_i, \vec{\alpha}) &= \bar{F}_1\left(b_i, \bar{F}(\lfloor \frac{1}{2}b_i \rfloor, \vec{\alpha}), \hat{F}(\lfloor \frac{1}{2}b_i \rfloor, \vec{\alpha}), \vec{\alpha}\right) \\ \bar{F}(b_i, \vec{\alpha}) &\leq t_0(b_i, \vec{\alpha}) \\ \text{and} \\ \hat{F}(0, \vec{\alpha}) &= \hat{F}_0(\vec{\alpha}) \\ \hat{F}(b_i, \vec{\alpha}) &= \hat{F}_1\left(b_i, \bar{F}(\lfloor \frac{1}{2}b_i \rfloor, \vec{\alpha}), \hat{F}(\lfloor \frac{1}{2}b_i \rfloor, \vec{\alpha}), \vec{\alpha}\right) \\ \hat{F}(b_i, \vec{\alpha}) &\leq t_1(b_i, \vec{\alpha}, \bar{F}(b_i, \vec{\alpha}))\end{aligned}$$

By MLRN such definition is correct.

Now we can apply the induction rule for open formulas and bounded existential introduction rule to get a  $(\text{QF}(\mathcal{B}) - \text{PIND})$  derivation of  $\Gamma(\vec{\alpha}), \psi(t(\vec{\alpha}), \vec{\alpha})$ . The claim follows now from the inductive hypothesis.

**Theorem 6** *The class of provably total functions of  $(\Sigma_1^b - \text{PIND})$  is exactly the class of the  $\mathcal{B}$  functionals.*

**Proof:**

As in the first order case one first shows that if  $(\Sigma_1^b - \text{PIND}) \vdash (\forall \vec{\alpha}) \exists y \leq t(\vec{\alpha}) \phi(\vec{\alpha}, y)$  where  $\phi$  is a  $\Sigma_1^b$  formula, then for some  $n$ ,  $(n - \Sigma_1^b) - \text{PIND} \vdash (\forall \vec{\alpha}) (\exists y \leq t(\vec{\alpha})) \phi(\vec{\alpha}, y)$  and we can also assume that  $\phi$  is a strict  $\Sigma_1^b$  formula of the language which includes a symbol for every function from  $\mathcal{B}$ . Then by the previous Lemma  $(\text{QF}(\mathcal{B}) - \text{PIND}) \vdash (\forall \vec{\alpha}) (\exists y \leq t(\vec{\alpha})) \phi(\vec{\alpha}, y)$ . Now the claim of the Theorem follows from the corresponding fact about  $(\text{QF}(\mathcal{B}) - \text{PIND})$  (whose proof is identical to the first order case). ■

The following lemma is a crucial tool in our proofs.

**Lemma 7** *Let  $P(|f|, |\vec{x}|)$  be a second order polynomial of depth  $p$  and  $t^{\vec{P}} = t_0^P(\vec{x}), \dots, t_p^P(f, \vec{x}, z_1, \dots, z_p)$  the corresponding sequence of terms in variables  $\vec{z}^P = z_1^P, \dots, z_p^P$  associated with the polynomial  $P$ . Then*

$$\begin{aligned}\mathbf{R}_2^1 \vdash & (\forall f)(\forall \vec{x})(\exists u)(\forall z_1 \leq t_0^P(\vec{x})) \dots (\forall z_p \leq t_{p-1}^P(f, \vec{x}, z_1, \dots, z_{p-1})) \\ & (|t_p^P(f, \vec{x}, z_1, \dots, z_p)| \leq |u|)\end{aligned}$$

Thus, despite the fact that second order polynomials are not basic feasible, they are provably bounded even in the theory  $\mathbf{R}_2^1$ . Notice that the reason why, of course, we cannot conclude that second order polynomials are feasible functionals from the above provability layes in the fact that formula which defines the graph of the functional is not a  $\Sigma_1^b$  formula but a conjunction of a  $\Sigma_1^b$  and



a  $\Pi_1^b$  formula:

$$((\forall z_1 \leq t_0^P(\vec{x})) \dots (\forall z_p \leq t_{p-1}^P(f, \vec{x}, z_1, \dots, z_{p-1}))(t_p^P(f, \vec{x}, z_1, \dots, z_p) \leq u)) \wedge ((\exists z_1 \leq t_0(\vec{x})) \dots (\exists z_d \leq t_{d-1}(f, \vec{x}, z_1, \dots, z_{d-1}))(t_d(f, \vec{x}, z_1, \dots, z_d) = u)))$$

**Proof:** Assume the opposite and fix  $f, \vec{x}$  such that for the formula

$$\Psi(v, f, \vec{x}) \equiv (\exists z_1 \leq t_0^P(\vec{x})) \dots (\exists z_p \leq t_{p-1}^P(f, \vec{x}, z_1, \dots, z_{p-1})) (|t_p^P(f, \vec{x}, z_1, \dots, z_p)| \geq v) \quad (25)$$

we have  $(\forall u)\Psi(|u|, f, \vec{x})$ . Fix now an arbitrary  $v$  and consider the formula  $\Psi^*(i, v, f, \vec{x}) \equiv \Psi(v \upharpoonright i, f, \vec{x})$ . Then clearly  $\Psi^*(0, v, f, \vec{x})$  holds. Assume now that  $\Psi^*(i, v, f, \vec{x})$  holds; then for some  $\underline{z}_1, \dots, \underline{z}_{p-1}$  we have  $(|t_p^P(f, \vec{x}, \underline{z}_1, \dots, \underline{z}_{p-1})| \geq v \upharpoonright i)$ . Let  $c_i = t_p^P(f, \vec{x}, \underline{z}_1, \dots, \underline{z}_{p-1})$ ; then  $v \upharpoonright i \leq c_i$ . But then  $|v \upharpoonright (2i)| \leq 2|v \upharpoonright i| = |(v \upharpoonright i)^2|$  and consequently  $v \upharpoonright (2i) \leq 2(v \upharpoonright i)^2 \leq 2|c|^2 = |(c\#c)^2|$ . Thus,  $(\exists d \leq (c\#c)^2)(v \upharpoonright (2i) = |d|)$ ; since  $(\forall u)\Psi(|u|, f, \vec{x})$  we get  $\Psi(v \upharpoonright (2i), f, \vec{x})$ , i.e.  $\Psi^*(2i, v, f, \vec{x})$ . The above argument shows that for the  $\Sigma_1^b$  formula  $\Psi^*$  the following holds:

$$\Psi^*(0, v, f, \vec{x}) \wedge (\forall i)(\Psi^*(i, v, f, \vec{x}) \rightarrow \Psi^*(2i, v, f, \vec{x})).$$

Using the corresponding instance of the  $\Sigma_1^b$ -LPIND, we get  $(\forall i)\Psi^*(v, |i|, f, \vec{x})$ . Thus,  $\Psi^*(v \upharpoonright |v|, f, \vec{x})$ , i.e.  $\Psi(v, f, \vec{x})$ . Consequently,  $(\exists z_1 \leq t_0^P(\vec{x})) \dots (\exists z_p \leq t_{p-1}^P(f, \vec{x}, z_1, \dots, z_{p-1}))(|t_p^P(f, \vec{x}, z_1, \dots, z_p)| \geq v)$  holds; since  $v$  was an arbitrary parameter, we get

$$(\forall v)(\exists w)(v = |w|). \quad (26)$$

Property (26) and  $\Sigma_1^b$ -LPIND imply that  $\Sigma_1^b$ -PIND and consequently also  $\Sigma_1^b$ -LIND both hold. (In fact, using (26) again, we can get that  $\Sigma_1^b$ -IND and  $\Pi_1^b$ -IND also hold.) We now show that for all  $k \leq p$  the following holds:

$$(\exists u_k)(\forall z_1 \leq t_0^P(\vec{x})) \dots (\forall z_k \leq t_{k-1}^P(f, \vec{x}, z_1, \dots, z_{k-1}))(t_k^P(f, \vec{x}, z_1, \dots, z_k) \leq u_k) \quad (27)$$

For  $k = 0$ , using (26), we find  $w_0$  such that  $t_0^P(\vec{x}) = |w_0|$  and then apply  $\Pi_1^b$ -LIND on  $y$  in the formula

$$(\exists z_0^* \leq |w_0|)(\forall z_0 \leq y)(t_1^P(f, \vec{x}, z_0) \leq t_1^P(f, \vec{x}, z_0^*)) \quad (28)$$

to get  $(\exists z_0^* \leq |w_0|)(\forall z_0 \leq |w_0|)(t_1^P(f, \vec{x}, z_0) \leq t_1^P(f, \vec{x}, z_0^*))$ , i.e.  $(\exists z_0^* \leq t_0^P(\vec{x}))(\forall z_0 \leq t_0^P(\vec{x}))(t_1^P(f, \vec{x}, z_0) \leq t_1^P(f, \vec{x}, z_0^*))$ . We now take  $u_0 = t_1^P(f, \vec{x}, z_0^*)$ .

Similarly, assuming that

$$(\exists u_k)(\forall z_1 \leq t_0^P(\vec{x})) \dots (\forall z_k \leq t_{k-1}^P(f, \vec{x}, z_1, \dots, z_{k-1}))(t_k^P(f, \vec{x}, z_1, \dots, z_k) \leq u_k)$$

and taking  $w_k$  such that  $|w_k| = u_k$  we prove by induction

$$(\exists z_{k+1}^* \leq |w_k|)(\forall z_{k+1} \leq |w_k|)(t_{k+1}^P(f, \vec{x}, z_{k+1}) \leq t_{k+1}^P(f, \vec{x}, z_{k+1}^*)).$$

Taking  $u = t_{k+1}^P(f, \vec{x}, z_{k+1}^*)$  we get a contradiction with (25).  $\blacksquare$

We can now prove our main results.

**Theorem 8** Assume that the functional  $F(f, \vec{x}, y)$  is defined from the functionals  $G(f, \vec{x})$  and  $H(f, \vec{x}, z, y)$  by polynomially bounded recursion on notation with the bound  $Q(|f|, |\vec{x}|, |y|)$ . Then  $F(f, \vec{x}, y)$  is a basic feasible functional.

**Proof:** Just replace second order polynomial bound with an appropriate  $\Sigma_1^b$  formula. Then a simple application of  $\Sigma_1^b$ -LIND proves that such a  $\Sigma_1^b$  formula defines a total function. ■

**Theorem 9** Assume that the functional  $F(f, \vec{x})$  is defined from the functionals  $G$  and  $H$  by polynomially bounded recursion of polynomial length with bounds  $(Q, P)$ . Then the functional  $F(f, \vec{x})$  is a basic feasible functional.

**Proof:** Assume that  $P(|f|, |\vec{x}|)$  and  $Q(|f|, |\vec{x}|)$  are two second order polynomials and let  $G(f, \vec{x})$  and  $H(f, \vec{x}, y, z)$  be two basic feasible functionals. Let the functional  $F^*(f, \vec{x}, y)$  satisfies

$$\begin{aligned} F^*(f, \vec{x}, 0) &= G(f, \vec{x}) \\ F^*(f, \vec{x}, y + 1) &= H(f, \vec{x}, F^*(f, \vec{x}, y), y), \\ (\forall y \leq P(|f|, |\vec{x}|)) &(|F^*(f, \vec{x}, y)| \leq Q(|f|, |\vec{x}|)) \\ (\forall y)(y \geq P(|f|, |\vec{x}|)) &\rightarrow F^*(f, \vec{x}, y) = F^*(f, \vec{x}, P(|f|, |\vec{x}|)) \\ (\forall f)(\forall f^*)((\forall y \leq P(|f|, |\vec{x}|)) &(F^*(f, \vec{x}, y) = F^*(f^*, \vec{x}, y)) \rightarrow \\ &(\forall y)(F^*(f, \vec{x}, y) = F^*(f^*, \vec{x}, y))) \end{aligned}$$

and that the functional  $F(f, \vec{x})$  is defined by

$$F(f, \vec{x}) = F^*(f, \vec{x}, P(|f|, |\vec{x}|)). \quad (29)$$

We replace the functional  $F(f, \vec{x})$  by  $\hat{F}(f, \vec{x})$  defined as follows. At each recursive stage  $u + 1$  of the computation of  $F^*(f, \vec{x}, u + 1)$  we also evaluate an approximation  $P^*(|f|, |\vec{x}|, u)$  of  $P(|f|, |\vec{x}|)$  on the basis of the values  $f(z_i)$  which were used in the computations of  $F^*(f, \vec{x}, z)$ , for  $z \leq u$ . Thus  $P^*(|f|, |\vec{x}|, u) = P(|f_u^*|, |\vec{x}|)$  where  $f_u^*(z_i) = f(z_i)$  for all  $z_i$  such that  $f(z_i)$  is used in the computations of  $F^*(f, \vec{x}, z)$ ,  $z \leq u$ , and  $f_u^*(w) = 0$  otherwise.

If  $u + 1$  is larger than  $P^*(|f|, |\vec{x}|, u)$  the recursion is aborted and the value of  $F^*(f, \vec{x}, w)$  for all  $w > u$  is set equal to the value  $F^*(f, \vec{x}, u)$ .

**Claim:** Functional  $\hat{F}(f, \vec{x})$  defined above has the same values as  $F(f, \vec{x})$ .

To prove the claim assume  $u + 1$  is (the first value) larger than  $P^*(|f|, |\vec{x}|, u)$ . Then  $P^*(|f_u^*|, |\vec{x}|, u) = P(|f_u^*|, |\vec{x}|)$  and so for  $f_u^*$  the value of the approximation  $P^*(|f_u^*|, |\vec{x}|, u)$  is equal to the value of the second order polynomial  $P(|f_u^*|, |\vec{x}|)$ . However, using the condition

$$(\forall y)(y \geq P(|f_u^*|, |\vec{x}|)) \rightarrow F^*(f_u^*, \vec{x}, y) = F^*(f_u^*, \vec{x}, P(|f_u^*|, |\vec{x}|))$$

we get that the recursion procedure produces for all  $y \geq P(|f_u^*|, |\vec{x}|)$  the same value  $F^*(f_u^*, \vec{x}, P(|f_u^*|, |\vec{x}|))$ . Using the condition

$$\begin{aligned} (\forall f)(\forall f^*)((\forall y \leq P(|f^*|, |\vec{x}|)) &(F^*(f, \vec{x}, y) = F^*(f^*, \vec{x}, y)) \rightarrow \\ &(\forall y)(F^*(f, \vec{x}, y) = F^*(f^*, \vec{x}, y))) \end{aligned}$$

we get that indeed

$$(\forall y)(F^*(f, \vec{x}, y) = F^*(f_u^*, \vec{x}, y))$$

This implies that

$$(\forall y)(y \geq P^*(|f_u^*|, |\vec{x}|) \rightarrow F^*(f, \vec{x}, y) = F^*(f_u^*, \vec{x}, P^*(|f_u^*|, |\vec{x}|)))$$

Since  $P(|f|, |\vec{x}|) \geq P^*(|f_u^*|, |\vec{x}|)$  we get that

$$F^*(f, \vec{x}, P(|f|, |\vec{x}|)) = F^*(f_u^*, \vec{x}, P^*(|f_u^*|, |\vec{x}|))$$

. However, the computation of  $F^*(f_u^*, \vec{x}, P^*(|f_u^*|, |\vec{x}|))$  clearly satisfies definition of the computation of  $\hat{F}(f, \vec{x})$ , which proves our claim.

We now formalize in  $(\Sigma_1^b - PIND)$  definition of  $\hat{F}(f, \vec{x})$ . We encode the computations for the recursion steps and for approximations  $P^*$  of the second order polynomial  $P$  on the basis of values  $f(z_i)$  used in the previous steps of recursion. Using the fact that

$$(\forall y \leq P(|f|, |\vec{x}|))(F^*(f, \vec{x}, y) \leq Q(|f|, |\vec{x}|))$$

and that  $(\forall y)(y \geq P(|f|, |\vec{x}|) \rightarrow F^*(f, \vec{x}, y) = F^*(f, \vec{x}, P(|f|, |\vec{x}|)))$  we can replace second order polynomial bound  $Q(|f|, |\vec{x}|)$  with a  $\Sigma_1^b$  formula with the existential prefix  $\exists \vec{z} \leq \vec{t}^Q$  with terms  $\vec{t}$  associated with the polynomial  $Q$ , which defines the graph of  $\hat{F}(f, \vec{x})$ .

By Lemma (7) it is provable in  $(\Sigma_1^b - PIND)$  that there exists  $u$  such that  $u + 1$  is larger than  $P^*(|f|, |\vec{x}|, u)$  and so the computation of  $\hat{F}(f, \vec{x})$  eventually terminates. This implies that  $\hat{F}(f, \vec{x})$  is provably total in  $(\Sigma_1^b - PIND)$  and thus a basic feasible functional. By our claim so is  $F(f, \vec{x})$ . ■

We can now improve the Cook Kapron result on machine models. Recall that a functional  $F(\vec{f}, \vec{x})$  is *computable in polynomial time* if there exists an oracle Turing machine  $M$  with oracles for functions  $\vec{f}$  and a second order polynomial  $P(|\vec{f}|, |\vec{x}|)$  such that  $M$  computes  $F(\vec{f}, \vec{x})$  and for all  $\vec{f}, \vec{x}$ , the running time  $T(\vec{f}, \vec{x})$ , obtained by counting each oracle query as a *single step* regardless of the size of the oracle output, satisfies

$$T(\vec{f}, \vec{x}) \leq P(|\vec{f}|, |\vec{x}|).$$

**Theorem 10** *A functional  $F(\vec{f}, \vec{x})$  is a polynomial time computable functional if and only if it is a basic feasible functional.*

**Proof:** By an easy inductive argument on the complexity of the definition of functionals one can show that every basic feasible functional is polynomial time computable (This holds even if we count every oracle query needed to get  $f(z)$  as  $|f(z)|$  many steps of computation, as it is proved in see [5]). Assume that for a second order polynomial  $P(|f|, |\vec{x}|)$  a Turing machine halts in

$P(|f|, |\vec{x}|)$  many steps. Then at each stage of the computation the working tape and the oracle input tape can contain at most  $P(|f|, |\vec{x}|)$  many symbols, and so the oracle output tape can contain at most  $|f|(P(|f|, |\vec{x}|))$  many symbols. It is easy to see that this implies that the output of such a machine can be obtained by polynomially bounded recursion of polynomial length, and thus it is a basic feasible functional. ■

## 4 The Bibliography

### References

- [1] S. R. Buss: Bounded Arithmetic, Bibliopolis, 1986.
- [2] P. Clote, G. Takeuti: Bounded Arithmetic for NC, ALogTIME, L and NL, *Annals of pure and applied logic*, 56, (1992).
- [3] P. Clote, A. Ignjatovic, B. Kapron: Parallel computable higher type functionals, *34th Annual Symposium on Foundations of Computer Science*, November 1993.
- [4] A. Cobham: The intrinsic computational difficulty of functions In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science II*, pp. 24-30. North-Holland, 1965.
- [5] B. Kapron, S. Cook: A new characterization of Mehlhorn's poly time functionals. In *Proceedings of the 32nd Symposium on Foundations of Computer Science*, 1991, pp. 342-347.
- [6] A. Ignjatovic: hand-written notes
- [7] B. M. Kapron: *Feasible Computations in Higher Types*, Ph.D. Thesis, University of Toronto, 1991.
- [8] B. Allen: Arithmetizing Uniform NC, *Annals of pure and applied logic*, 53, (1991)
- [9] K. Mehlhorn: Polynomial and abstract subrecursive classes. JCSS, 12:147-178.
- [10] W. Sieg: Fragments of Arithmetic, *Annals of Pure and Applied Logic* 28 (1985), pp. 33-71, North Holland.
- [11] W. Sieg: Herbrand Analyses, *Archive for Mathematical Logic* 30 (1991), pp. 409-441, Springer-Verlag.
- [12] G. Takeuti: Proof Theory, Second edition, North-Holland, 1987.

- [13] M. Townsend: Complexity for type-2 relations. *Notre Dame Journal of Formal Logic*, 31:241-262,1990.