Open least element principle and bounded query computation

L. D. Beklemishev *

Steklov Mathematical Institute
Gubkina 8, 117966 Moscow, Russia
e-mail: `bekl@mi.ras.ru`, `beklemi@math.uni-muenster.de`

Abstract. We show that elementary arithmetic formulated in the language with a free function symbol f and the least element principle for open formulas (where we assume that the symbols for all elementary functions are included in the language) does not prove the least element principle for bounded formulas in the same language. A related result is that composition and any number of unnested applications of bounded minimum operator are, in general, insufficient to generate the elementary closure of a function, even if all elementary functions are available. Thus, unnested bounded minimum operator is weaker than unnested bounded recursion.

1 Introduction and motivation

This paper arose out of the problem of separating the schemes of Δ_1-induction and Σ_1-collection in arithmetic [4, 6]. It turns out that this question is closely related to the comparison of different operators generating the elementary closure of a class of functions and to the problems of separating the corresponding systems of subrecursive arithmetic. These questions are also natural from a purely computational point of view.

In this paper we compare the relative strength of bounded μ-operator and bounded recursion. We show that unnested bounded μ-operator is, in general, weaker than unnested bounded recursion. In contrast, it is well known that, when nestings are allowed, each of the two operators together with composition is sufficient to generate the elementary closure of a class of functions.

We compare the strength of the two operators against the problem of computing the maximum of a function on a finite interval. In order to compute $\max_{i\leq x} f(i)$ on a Turing machine with a function oracle for f one needs of order x different oracle queries (see [2]). In particular, any Turing machine that may only ask a bounded number of queries cannot, in general, compute this function. This implies that the class of functions generated from all elementary functions and f by composition does not, in general, coincide with the elementary closure of a function f. This idea was used in [2] to show the independence of Σ_1-collection rule in arithmetic, which improved the result of Parsons on the independence of

* Supported by Alexander von Humboldt Foundation and RFBR grant 98-01-00249.

Σ_1-collection schema from the set of all true arithmetical Π_2-sentences (see also [8] for a related work on subrecursive degrees).

If unnested applications of bounded μ-operator are allowed on a par with composition, then the available power of computation increases compared to the bounded query oracle Turing machine. It is worth explaining here informally, why such a computation mechanism is still too weak to compute the maximum of f.

A μ-operator of the form $\mu i \leq x.\ R(i)$, where $R(i)$ is a bounded query predicate, can be interpreted in terms of a parallel bounded query machine. To evaluate this operator the machine generates $x+1$ independent subprocesses, i-th process P_i evaluates the predicate $R(i)$ and returns **true** or **false**. Then the machine runs through their outputs to find the least i such that $R(i)$ evaluates to **true**.

Since $R(x)$ is a bounded query predicate, all the subprocesses have a uniform bound on the number of queries each of them may ask. More important still, each subprocess P_i only returns **true** or **false**, that is, exactly one bit. This means that the processes cannot exchange too much information. This is crucial for the fact that $\max_{i \leq x} f(i)$ cannot be computed by such a machine: each subprocess P_i can only learn the values of f on a boundedly small subset of the large interval $[0, x]$, but it lacks the ability to communicate, say, the maximum of these values to other processes. (Compare with the usual algorithm of computing the maximum of f by querying successively $f(0), f(1), \ldots, f(x)$. Here, one has to always store the intermediate maximum value of f, which may potentially exceed any bounded number of bits.)

Of course, this rough idea will be made more precise in the proof of our main result. This proof is based on a recursion-theoretic diagonal construction that involves a combinatorial argument using infinite Ramsey theorem.

2 Statement of the results

As usual, for a given predicate $R(x, \boldsymbol{v})$ the expression $\mu x \leq a.\ R(x, \boldsymbol{v})$ denotes the function

$$m(a, \boldsymbol{v}) = \begin{cases} \text{the minimal } x \leq a \text{ such that } R(x, \boldsymbol{v}) \text{ holds,} & \text{if } \exists x \leq a R(x, \boldsymbol{v}) \\ a+1, & \text{otherwise.} \end{cases}$$

For a set of functions K, let $\mathbf{C}(K)$ denote the closure of K and the class of elementary functions $\mathcal{E}$ under composition. Further, let $[K, \mathbf{M}]$ denote the closure of the class $K \cup \mathcal{E}$ under composition and unnested applications of bounded minimum operator, that is, the closure under composition of $K \cup \mathcal{E}$ and all functions of the form $\mu x \leq a.\ R(x, \boldsymbol{v})$, where $R(x, \boldsymbol{v}) \in \mathbf{C}(K)$. Similarly, $[K, \mathbf{BR}]$ denotes the closure of $K \cup \mathcal{E}$ under composition and unnested applications of bounded recursion schema, that is, primitive recursion bounded by some function from $\mathbf{C}(K)$. The closure of $K \cup \mathcal{E}$ under composition and (nested) bounded recursion is called the *elementary closure* of K and is denoted $\mathbf{E}(K)$. By a result of Parsons [9], $\mathbf{E}(f) = \mathbf{C}(\bar{f})$, where $\bar{f}$ denotes the function $\bar{f}(x) = \langle f(0), \ldots, f(x) \rangle$.

It is known [10] that $\mathbf{E}(K)$ coincides with the closure of $K \cup \mathcal{E}$ under composition and (nested) bounded minimum operator. It is also easy to see from [10] that $\mathbf{E}(K) = [K, \mathbf{BR}]$. On the other hand, we prove the following theorem.

Theorem 1. *There is a function $f : \mathbb{N} \to \mathbb{N}$ such that*

$$[\mathbf{C}(f), \mathbf{M}] \neq [\mathbf{C}(f), \mathbf{BR}] = \mathbf{E}(f).$$

This means that unnested bounded μ-operator is insufficient to generate the elementary closure of a function. In contrast, we also show that for any f, $[[\mathbf{C}(f), \mathbf{M}], \mathbf{M}] = \mathbf{E}(f)$.

Our main concern will be the counterpart of these results in formal arithmetic. We consider the language of first order Kalmar elementary arithmetic EA (with function symbols for all elementary functions and the only relation symbols $\leq$ and $=$) enriched by a new unary function symbol f. The set of open formulas in this language will be denoted $\mathbf{C}(f)$. The set of bounded formulas, that is, the formulas obtained from atomic ones using boolean connectives and bounded quantifiers (f may occur in bounding terms) is denoted $\Delta_0(f)$. Relativized elementary arithmetic $\mathsf{EA}(f)$ is the extension of EA in the enriched language without any additional mathematical axioms for f.

We study the comparative strength of the *least element principle* for open and for bounded formulas. For a formula $R(x)$, let L_R denote the following formula:

$$R(a) \to \exists x \leq a \, (R(x) \wedge \forall y < x \, \neg R(y)).$$

Of course, we assume that $R(x)$ may involve other parameters apart from x. $L\mathbf{C}(f)$ will denote the set of formulas L_R for all open R; $L\Delta_0(f)$ is similarly defined. By abusing the terminology, $L\mathbf{C}(f)$ will also denote the theory axiomatized over $\mathsf{EA}(f)$ by this schema. The related *induction schema* is similarly defined: I_R denotes the formula

$$R(0) \wedge \forall x \leq a \, (R(x) \to R(x+1)) \to R(a),$$

$I\mathbf{C}(f)$ denotes the set of formulas I_R for all open R, $I\Delta_0(f)$ denotes the set of formulas I_R for all bounded R.

Obviously, for any formula R, the formula $L_{\neg R}$ implies I_R, and $I_{\forall u \leq x \, \neg R(u)}$ implies L_R. This shows that the schemata $I\Delta_0(f)$ and $L\Delta_0(f)$ are equivalent over $\mathsf{EA}(f)$. Notice, however, that the formula $\forall u \leq x \, \neg R(u)$ involves bounded quantifiers. In fact, over $\mathsf{EA}(f)$ the induction schema $I\mathbf{C}(f)$ turns out to be strictly weaker than the corresponding least element principle $L\mathbf{C}(f)$. This fact is related to the question of separating the schema of arithmetical Δ_1-induction from the Σ_1-collection schema [4, 6]. We give a proof of this fact in a subsequent paper.

Our goal here is to prove the following result.

Theorem 2. $L\mathbf{C}(f) \nvdash L\Delta_0(f)$.

Before starting with the proof, let us notice that the theorem above is sensitive to the presence of additional axioms for f. For example, if the additional axioms expressing that f is any of the usual fast growing functions of the Grzegorczyk hierarchy are added to $L\mathbf{C}(f)$, then the schema $L\Delta_0(f)$ will be provable. In fact, an axiom asserting the monotonicity of f and an elementary definition of the graph of f suffice [1]. As a trivial example consider the axiom $\forall x\, f(x) = 0$. In this case both theories are equivalent to the elementary arithmetic EA.

The proof of Theorem 2 goes in two relatively independent steps. The first step is a proof-theoretic reduction of the question of separation of the two axiom systems to a purely computation-theoretic question. This will be a more or less straightforward application of Herbrand's theorem for $\exists\forall$ formulas. The second step is a recursion-theoretic construction, which allows to separate the classes of computable functions resulting from the previous Herbrand analysis. This result strengthens Theorem 1 and is obtained by a slight modification of its proof.

Applications of Herbrand's theorem for $\exists\forall$ formulas have proved to be very useful in the questions of separating the systems of bounded arithmetic. Connections between the classes of functions representable in systems of bounded arithmetic and bounded query computation have also been established (see [7]). This paper shows that a similar methodology is useful in the context of subrecursion and subrecursive arithmetic. However, there are some notable differences. An essential feature of our techniques is that we deal with function oracles, whereas bounded query computation has mostly been considered for the set oracle Turing machines. This difference seems to be essential, e.g., the problem of computing the maximum of a function has little meaning for 0–1-valued functions.

3 Herbrand analysis

Recall that $\bar{f}$ denotes the function $\bar{f}(x) = \langle f(0), \dots, f(x)\rangle$, assuming some standard elementary coding of finite sequences of numbers. Notice that the graph of $\bar{f}$ can be defined in the language of $\mathsf{EA}(f)$ by the following bounded formula:

$$\bar{f}(x) \simeq y \equiv [y \in Seq \wedge lh(y) = x + 1 \wedge \forall i \leq x\ (y)_i = f(i)]. \tag{1}$$

The following lemma is straightforward, but we shall give a detailed argument in order to see how much induction is actually used. For $n \geq 1$ let $\Pi^b_n(f)$ and $\Sigma^b_n(f)$ denote the classes of formulas obtained from $\mathbf{C}(f)$ by n alternating blocks of bounded quantifiers, starting from $\forall$ and $\exists$, respectively. For technical reasons (that will only be essential in Section 5) we assume that all the bounding terms are elementary, that is, do not involve the function symbol f.

Lemma 1. *$L\Delta_0(f) \vdash \forall x \exists y \bar{f}(x) \simeq y$.*

Proof. By induction on x we first prove the following statement:

$$\forall x\, [\exists u \leq x\, \forall i \leq x\, f(i) \leq f(u)].$$

The formula in square brackets (let us denote it $\varphi(x)$) is bounded and, in fact, belongs to the class $\Sigma^b_2(f)$. Moreover, the induction step $\forall x\, (\varphi(x) \to \varphi(x+1))$

is obviously provable in $\mathsf{EA}(f)$. Hence, this argument can be done in $I\Sigma^b_2(f)$. Further, taking $y = f(u)$ we can conclude within that theory and within $L\Delta_0(f)$ that $y = \max_{i \le x} f(i)$ exists.

As a separate argument we now prove the following statement:

$$\forall i \le x\ f(i) \le y \to \exists v \le b(x, y)\ \bar{f}(x) \simeq v,$$

for a suitable elementary term $b(x, y)$ that bounds the code of a sequence given its length and a bound to its elements. This statement follows by the least element principle, for $\bar{f}(x)$ is the least element $v \le b(x, y)$ such that $\forall i \le x\ f(i) \le (v)_i$. (This holds under the usual conventions on the coding of sequences.) Thus, this part of the argument is formalizable in $L\Pi^b_1(f)$.

In Section 5 we improve on the first part of the above argument and show that the statement $\forall x \exists y \bar{f}(x) \simeq y$ is provable in $L\Pi^b_1(f)$. In fact, $L\Pi^b_1(f)$ is shown to be equivalent to $L\Delta_0(f)$. Our main aim in this paper is to show that the totality of $\bar{f}$ is *not* provable in $L\mathbf{C}(f)$.

The following easy lemma states that symbols for all functions of the class $[\mathbf{C}(f), \mathbf{M}]$ can be introduced in a definitional extension of the theory $L\mathbf{C}(f)$. First, for any formula $R(x) \in \mathbf{C}(f)$ we introduce a symbol m_R for the function $\mu x \le a.\ R(x)$ (here and below we ignore possible additional parameters in R and m_R). Second, let M_R denote the following formula:

$$R(a) \to (R(m_R(a)) \wedge \forall y < m_R(a)\ \neg R(y)).$$

Let $L\mathbf{C}(f)^\circ$ denote the extension of $\mathsf{EA}(f)$ by symbols $m_R(x)$ for all $R \in \mathbf{C}(f)$ and the corresponding axioms M_R.

Lemma 2. *$L\mathbf{C}(f)^\circ$ is a conservative definitional extension of $L\mathbf{C}(f)$.*

Proof. The two things to notice is that axioms M_R logically imply the corresponding axioms L_R (because M_R implies $R(a) \to m_R(a) \le a$), and that L_R also implies that the minimum is unique.

Notice that $L\mathbf{C}(f)^\circ$ has a purely universal axiomatization. A version of Herbrand's theorem for $\exists\forall$ formulas suitable for our present purposes reads as follows [3].

Lemma 3. *Let T be a theory axiomatized by a set of purely universal formulas, and let $\varphi(x, u, y)$ be an open formula such that $T \vdash \exists y \forall u \varphi(x, u, y)$. Then there are terms $t_0(x)$, $t_1(x, u_0)$, ..., $t_n(x, u_0, \dots, u_{n-1})$ such that*

$$T \vdash \varphi(x, t_0(x), u_0) \vee \varphi(x, t_1(x, u_0), u_1) \vee \ldots \vee \varphi(x, t_n(x, u_0, \dots, u_{n-1}), u_n).$$

We now apply this theorem to the formula $\exists y \bar{f}(x) \simeq y$. In other words, we take T to be $L\mathbf{C}(f)^\circ$ and $\varphi(x, u, y)$ to be $(u \le x \to f(u) = (y)_u)$ and obtain the following corollary.

Corollary 1. *If* $LC(f) \vdash \forall x \exists y\, \bar{f}(x) \simeq y$*, then there are terms* $t_0(x)$*,* $t_1(x, u_0)$*,* $\dots$*,* $t_n(x, u_0, \dots, u_{n-1})$ *in* $LC(f)^\circ$ *such that the following disjunction holds for any* f *and all* $x, u_0, \dots, u_n \in \mathbb{N}$ *(and is, in fact, provable in* $LC(f)^\circ$*):*

$$\begin{aligned}
&(u_0 \le x \to f(u_0) = (t_0(x))_{u_0}) \quad \vee \\
&(u_1 \le x \to f(u_1) = (t_1(x, u_0))_{u_1}) \quad \vee \quad \cdots \\
&(u_n \le x \to f(u_n) = (t_n(x, u_0, \dots, u_{n-1}))_{u_n}).
\end{aligned} \tag{2}$$

In order to compute $\bar{f}(a)$ we do not need to know any values of f outside the interval $[0, a]$. This idea is captured in the following definition of *a-reduced* $[\mathbf{C}(f), \mathbf{M}]$-term t, where a is a distinguished free variable of t.

A term t is *a-reduced*, if every term s such that $f(s)$ is a subterm of t graphically has the form $s = \min(s_0, a)$, for some term s_0. (We also consider the subterms $f(s)$ occurring inside the μ-operators $\mu i \le t_0.\ R(i)$, that is, in t_0 and R.)

The following corollary allows us to restrict our attention below to the reduced terms.

Corollary 2. *In the formulation of Corollary 1 we can assume all the terms* $t_0(x)$*,* $t_1(x, u_0)$*,* $\dots$*,* $t_n(x, u_0, \dots, u_{n-1})$ *to be x-reduced.*

Proof. Let the terms t_i for $i < n$ satisfy the conclusion of Corollary 1. Replace all subterms of the form $f(s)$ occurring in one of the terms t_i by the terms $f(\min(s, x))$. (We also make the substitution inside the subterms of the form $\mu z \le t.\ R(z)$, that is, in t and R.) Each term of the resulting sequence $t'_0(x)$, $t'_1(x, u_0), \dots, t'_n(x, u_0, \dots, u_{n-1})$ is x-reduced. We claim that this sequence also satisfies the conclusion of Corollary 1.

Indeed, the sequence of terms $t_0(x), t_1(x, u_0), \dots, t_n(x, u_0, \dots, u_{n-1})$ is supposed to satisfy the disjunction (2) for any function f. Let the function f' coincide with f on the interval $[0, x]$ and let $f'(y) = f(x)$ for $y > x$. Then, obviously, $\bar{f}'(x) = \bar{f}(x)$. Moreover, the value of $f'(s)$ for any s coincides with $f(\min(s, x))$. Hence the value of any term t'_i for f coincides with the value of t_i for f' (easy induction on the build-up of t_i). It follows that the sequence t'_i satisfies the conclusion of the corollary.

The next section will be devoted to the proof of the fact that for a suitably chosen f a sequence of (x-reduced) terms satisfying (2) cannot exist. This will be preceded by a proof of Theorem 1.

4 Bounded query μ-programs

Every term $t \in \mathbf{C}(f)$ can be considered as a program with an oracle for a function f, which may only ask a bounded number of oracle queries (this number is bounded by the number of occurrences of the symbol f in t and does not depend on the input of the program). The running time is then bounded by an elementary function of the size of the input together with all the oracle answers.

For technical reasons it will be convenient for us to deal with terms from $\mathbf{C}(f)$ in the format of programs (this fixes a specific order of subterms of a given term).

A *simple program* R is a sequence of assignments of the following kind:

$$\begin{aligned} y_0 &:= f(t_0(\boldsymbol{x})); \\ y_1 &:= f(t_1(\boldsymbol{x}, y_0)); \\ &\dots \\ y_n &:= f(t_n(\boldsymbol{x}, y_0, \dots, y_{n-1})) \\ R &:= R(\boldsymbol{x}, y_0, \dots, y_n). \end{aligned}$$

Here $t_0, \dots, t_n$ are elementary terms, $\boldsymbol{x}$ are the input variables, and R is an elementary output predicate. (We shall only consider simple programs that compute predicates, that is, return **true** or **false**.) The number n is called the *length* of the program R, and y_i are the *inner variables* of R.

Execution of a program for a given oracle function $f : \mathbb{N} \to \mathbb{N}$ is defined in the obvious way. If f is only a partial function, then the result of the execution of a program may not be defined (this happens, if the value of some term t_i computed during the execution of the program does not belong to $dom(f)$). A program R is *a-reduced*, if all the terms t_i of R have the form $\min(t_i', a)$, for some elementary term t_i', where a is a fixed input variable.

Clearly, any predicate $R \in \mathbf{C}(f)$ can be rewritten as a simple program. Moreover, an a-reduced predicate is represented as an a-reduced program.

Suppose R is a simple program of length n, and let the values of all input variables be fixed. Then the output of R can be considered as a boolean-valued function of the oracle answers obtained during its execution. However, there may be some dependencies between these answers. We will show that, under certain conditions on the values of f, these dependencies can be simplified. This is formally expressed by Lemma 4 below. The idea behind it is that by a suitable restriction of the (infinite) range of f, and hence of the range of possible oracle answers, one can insure that the output of any given simple program does not actually depend on the those answers. This idea already has a Ramsey-type flavour and, indeed, Ramsey theorem plays a central role in the argument. Technical details follow.

Let $f : D \to \mathbb{N}$ be a partial function with a finite domain D, and $Y \subseteq \mathbb{N}$ be an infinite set. Let $\prec$ be a linear ordering on $[0, n]$, it will be interpreted as a certain preference ordering of the inner variables $y_0, \dots, y_n$ of R. A sequence of numbers $\boldsymbol{c} = (c_0, \dots, c_n)$ from the interval $[0, a]$ and a sequence $(y_0, \dots, y_n)$ are *coherent* if they satisfy the following conditions:

f-constraints: $\forall i \leq n(c_i \in dom(f) \Rightarrow y_i = f(c_i))$;
$\boldsymbol{c}$-constraints: $\forall i, j \leq n(c_i = c_j \Rightarrow y_i = y_j)$.

If some of the following additional assumptions are satisfied, we speak about Y-, $\prec$- or $(Y, \prec)$-coherence.

Y-constraints: $\forall i \leq n(c_i \notin dom(f) \Rightarrow y_i \in Y)$;

$\prec$-constraints: If $i, j \leq n$ are such that $c_i, c_j \notin dom(f)$ and $(\forall k < i \ c_k \neq c_i)$ & $(\forall k < j \ c_k \neq c_j)$, then $i \prec j$ implies $y_i < y_j$.

Clearly, any coherent pair of sequences represents an extension f' of the given function f to the domain $D \cup \{c_0, \ldots, c_n\}$. Y- and $\prec$-constraints tell us that the values of this function are chosen in a pre-specified order from a given set Y. Given a sequence $\boldsymbol{c}$, one can define the subsequence of *essential* elements of $\boldsymbol{c}$ as follows. c_0 is essential if $c_0 \notin dom(f)$. c_{k+1} is essential if $c_{k+1} \notin dom(f)$ and for no $i \leq k$ do we have $c_i = c_{k+1}$. Note that the sequence of essential elements of $\boldsymbol{c}$ only depends on f. $\prec$-constraints state that the order of the values of f' on essential arguments agrees with the preference order $\prec$.

Notice the following obvious monotonicity property: if a pair of sequences is Y-coherent, then it is also Y'-coherent for any set $Y' \supseteq Y$.

Let a reduced program R and the values of the input variables of R be fixed. Let a be the value of the distinguished variable of R.

Lemma 4. *For any f, Y and $\prec$ as above, one can find an infinite subset $Y' \subseteq Y$ and a sequence $\boldsymbol{c} = (c_0, \ldots, c_n)$ such that any sequence $\boldsymbol{y} = (y_0, \ldots, y_n)$ $(Y', \prec)$-coherent with $\boldsymbol{c}$ yields an extension f' of f for which the program R terminates. Moreover, the values of the terms t_i computed during the execution of R on f' equal the constants c_i, and the output value of R is constant (either R is true for all such sequences $\boldsymbol{y}$, or false for all of them).*

Proof. We successively construct infinite sets $Y_0 \supseteq Y_1 \supseteq \ldots \supseteq Y_n$ and the corresponding numbers $(c_0, \ldots, c_n)$, so that the coherent values of f' taken from set Y_i on arguments $(c_0, \ldots, c_i)$ allow to execute the program R up to line i.

Since the values of the input variables are fixed, we simply define c_0 to be the value of t_0 and let Y_0 equal Y. Assume Y_k and $\boldsymbol{c} = (c_0, \ldots, c_k)$ are already constructed. Consider the term $t_{k+1}(y_0, \ldots, y_k)$. First, substitute in t_{k+1} the constants $f(c_i)$ for all variables y_i such that $c_i \in dom(f)$. Second, substitute for y_j the variable y_i, if c_i is essential, $c_j = c_i$ and $i < j$. Call the resulting term t'.

Obviously, the value of t_{k+1} on any sequence $\boldsymbol{y}$ coherent with $\boldsymbol{c}$ coincides with the value of t' on the subsequence of $\boldsymbol{y}$ corresponding to essential elements of $\boldsymbol{c}$.

The ordering $\prec$ of the essential elements of $\boldsymbol{c}$ allows us to naturally associate with t' a function

$$\varphi_{t'} : [Y_k]^p \to [0, a],$$

where p is the number of arguments of t' (and coincides with the number of essential elements of $\boldsymbol{c}$), and $[X]^p$ denotes the set of all p-element subsets of X.

The function $\varphi_{t'}$ is defined as follows. Let $e : [1, p] \to [0, k]$ enumerate the essential elements $(c_{e1}, \ldots, c_{ep})$ of $\boldsymbol{c}$. Let π be the unique permutation of $[1, p]$ such that

$$\pi i < \pi j \iff ei \prec ej,$$

that is, $e \circ \pi^{-1}$ enumerates the essential elements in the order induced by $\prec$. We define

$$t''(v_1, \ldots, v_p) = t'(v_{\pi 1}, \ldots, v_{\pi p}).$$

Any p-element subset $V \subseteq Y_k$ can be uniquely ordered into an increasing sequence $\boldsymbol{v}$ of elements of Y_k. We define $\varphi_{t'}(V) = t''(\boldsymbol{v})$.

By the infinite Ramsey theorem there is an infinite subset $Y_{k+1} \subseteq Y_k$ such that $\varphi_{t'}$ is constant on $[Y_{k+1}]^p$. Let c_{k+1} be the constant value of $\varphi_{t'}$ on $[Y_{k+1}]^p$. We show that $t_{k+1}(y_0, \dots, y_k) = c_{k+1}$ for all sequences $(y_0, \dots, y_k)$ which are $(Y_{k+1}, \prec)$-coherent with $\boldsymbol{c}$.

Indeed, let $(y_0, \dots, y_k)$ be $(Y_{k+1}, \prec)$-coherent with $\boldsymbol{c}$. Then

$$\begin{aligned} t_{k+1}(y_0, \dots, y_k) &= t'(y_{e0}, \dots, y_{ep}) \\ &= t''(y_{e\pi^{-1}0}, \dots, y_{e\pi^{-1}p}). \end{aligned}$$

By $\prec$-coherence we have

$$y_{e\pi^{-1}i} < y_{e\pi^{-1}j} \iff e\pi^{-1}i \prec e\pi^{-1}j \iff i < j.$$

This means that $(y_{e\pi^{-1}0}, \dots, y_{e\pi^{-1}p})$ is an increasing sequence, hence by the choice of Y_{k+1}

$$t''(y_{e\pi^{-1}0}, \dots, y_{e\pi^{-1}p}) = c_{k+1}.$$

Thus, we have constructed the required sequence $(c_0, \dots, c_n)$ and a set Y_n, which allow to execute R up to line n.

To ensure that the output value of R is constant we apply Ramsey theorem in the same fashion to the predicate $R(y_0, \dots, y_n)$ considered as a boolean valued function of $\boldsymbol{y} = (y_0, \dots, y_n)$. Then we obtain a subset $Y' \subseteq Y_n$ such that R is constant on any sequence $\boldsymbol{y}$ which is $(Y', \prec)$-coherent with $(c_0, \dots, c_n)$.

Lemma 5. *The set Y' can be chosen uniformly in $\prec$, that is, given f, Y and the values of input variables of R, one can point out a subset $Y' \subseteq Y$ that satisfies the conclusion of Lemma 4 for any preference ordering $\prec$.*

Proof. This follows from the fact that there are only finitely many possible orderings of $[0, n]$. Thus, we can enumerate all such orderings $\prec_0, \dots, \prec_s$. Applying Lemma 4 $s+1$ times we successively construct infinite sets $Y_0 \supseteq Y_1 \supseteq \dots \supseteq Y_s = Y'$. Then for any ordering $\prec_i$ one can point out a sequence $\boldsymbol{c}_i$ such that any $(Y', \prec_i)$-coherent sequence $\boldsymbol{y}$ is also $(Y_i, \prec_i)$-coherent with $\boldsymbol{c}_i$ (by the monotonicity property) and thus satisfies the conclusions of the previous lemma.

Next we define μ-programs. A *μ-operator* is an assignment of the following form:

$$y := \mu i \leq z.\ R(i, \boldsymbol{x}), \tag{$*$}$$

where R is a simple program with the input variables as shown. A μ-operator is a-reduced, if the simple program R is. If the variables $\boldsymbol{x}$ and z are already evaluated, then the output y of the μ-operator is defined, as usual, to be the minimal i for which $R(i, \boldsymbol{x})$ holds, if such an $i \leq z$ exists, and y equals $z + 1$, otherwise. A *μ-program* P is a sequence of simple assignments of the form $y := t(\boldsymbol{u})$ ($t(\boldsymbol{u})$ is an elementary term), $y := f(x)$ and of μ-operators $(*)$, which satisfies the following natural variable restrictions:

1) any inner variable is assigned its value in P only once;

2) for any occurrence of any of the above kinds of operators, $\boldsymbol{u}$, $\boldsymbol{x}$, z and x are either input variables or inner variables introduced earlier than y.

We will only consider μ-programs that are reduced in the sense that all the μ-operators are a-reduced, for a fixed input variable a. Obviously, such programs suffice to represent any a-reduced $[\mathbf{C}(f), \mathbf{M}]$-term.

Notice that there can be various possibilities of implementing μ-operator on different machines. A simple deterministic strategy would be to evaluate successively $R(0), R(1), \ldots, R(z)$ until the minimal i for which $R(i)$ evaluates to **true** is found. However, there is a possibility of implementing the μ-operator on a parallel machine, where we independently evaluate $R(0), R(1), \ldots, R(z)$ on different processors and then run through their single-bit outputs to find the minimal i for which $R(i)$ holds. Notice that these independent processors can be bounded in that each one of them may only ask a uniformly bounded number of oracle queries, whereas the simple deterministic strategy requires a potentially unbounded number of oracle queries. Another important restriction is that each of the processors only returns a single bit, that is, it cannot exchange too much information with other processors. This nondeterministic picture is useful for the understanding of our formal construction below.

Suppose we are given a μ-operator and the values of its input variables. Further, assume that an infinite set Y and a finite function f are given. We shall describe a procedure that allows to evaluate the μ-operator under some conditions on the choice of additional values of f . Simultaneously we extend f (adding no more than $n+1$ new elements to its domain), and go from Y to an (infinite) subset $Y' \subseteq Y$.

First, consider the simple program $R(x)$ for the input value $x = 0$. By Lemma 5 we obtain an infinite subset $Y_0 \subseteq Y$ such that any ordering $\prec$ of the inner variables of R uniquely determines a sequence $\boldsymbol{c}$ together with an output value of $R(0)$. If for one of the finitely many orderings $\prec$ the corresponding value equals **true**, then we evaluate μ to 0, set $Y' = Y_0 \setminus [0, \max(f)]$ and extend f to the set $\{c_0, \ldots, c_n\}$ by choosing a suitable tuple $\boldsymbol{y}$ $(Y', \prec)$-coherent with $\boldsymbol{c}$. (Here $\max(f)$ is the maximal value of f, recall that $dom(f)$ is finite.)

Otherwise (if all preference orderings yield the output value **false**), then we consider the input value $x = 1$ for R and construct the corresponding set $Y_1 \subseteq Y_0$. Again, μ is evaluated to 1, if at least one of the orderings yields value **true**, and to **false**, otherwise.

Proceeding in this way, we construct subsets $Y_0 \supseteq Y_1 \supseteq \ldots \supseteq Y_k$ for $k \leq z$. At the end, μ is evaluated to k, if a k is found for which $R(k)$ evaluates to **true** under this procedure, or μ is evaluated to $z+1$, otherwise. In the latter case we define $Y' = Y_z$ and $f' = f$.

Notice that as the result of running this procedure we evaluate the μ-operator and simultaneously define an infinite subset $Y' \subseteq Y$ and a function f' extending f. Obviously, this extension procedure preserves the injectivity of f.

Notice the following important property of this construction.

Lemma 6. *Let the evaluation procedure yield Y' and f', and let g be any total injective function extending f' such that $g(\mathbb{N} \setminus dom(f')) \subseteq Y'$. Then the value of the μ-operator computed for the function g coincides with the result of the above evaluation procedure.*

Proof. Consider the evaluation procedure. If μ was evaluated to 0, this means that $R(0)$ evaluated to **true** under some preference ordering. Yet, the essential arguments and values in that case are fixed, that is, $R(0)$ evaluates to **true** under f', and hence under g.

If μ was evaluated to 1, then $R(1)$ evaluates to **true** under f' and g for similar reasons. Let us execute the simple program $R(0)$ for g. Notice that by the construction of f', $f'(\mathbb{N} \setminus dom(f)) \subseteq Y'$. This means that all values assigned to the inner variables of R during the execution of $R(0)$ on g are elements of Y'. Moreover, the sequence of these values $\boldsymbol{y}$ and the corresponding arguments $\boldsymbol{c}$ of g satisfy f-, $\boldsymbol{c}$- and Y'-constraints.

Further, by our assumption g has different values on different arguments. Hence, the values y_i (for essential arguments c_i) are strictly linearly ordered. Let $\prec$ be any extension of the induced linear ordering to $[0, n]$ (it does not matter, how the inessential arguments are ordered). Then $\boldsymbol{y}$ and $\boldsymbol{c}$ will be $(Y', \prec)$-coherent. By the construction of Y', the value **false** of $R(0)$ is then uniquely determined for any $(Y', \prec)$-coherent sequences, that is, this value coincides with the computed value of $R(0)$ on g.

The argument showing that the values of $R(j)$ for $j \leq i$ are preserved, for the case that the μ-operator was evaluated to $i > 1$, is similar.

Now we show that the function $\bar{f}$ cannot be computed by a μ-program.

Lemma 7. *There is a total function $f : \mathbb{N} \to \mathbb{N}$ such that for any μ-program $P(x)$ there is a number a such that $P(a) \neq \bar{f}(a)$.*

Proof. First of all, by Corollary 2 it is sufficient to construct an f that falsifies any x-reduced μ-program (actually, we apply a particular case of this corollary for a sequence consisting of a single term t_0).

We enumerate all such programs $P_0, \ldots, P_\alpha, \ldots$ and construct f in such a way that by the stage α the function f is defined on a finite domain D_α and falsifies the programs $P_0, \ldots, P_{\alpha-1}$. We set $D_0 = \emptyset$.

Consider the program $P_\alpha(x)$. Let q be the total number of occurrences of the symbol f in P_α, where we also count the occurrences of f inside the μ-operators. Choose a number a so large that there are more than q elements in the set $[0, a] \setminus D_\alpha$. Evaluate $P_\alpha(a)$ along with constructing an infinite set $Y \subseteq \mathbb{N}$ and an extension of f according to the following rules.

Initialize $Y = \{x \in \mathbb{N} \mid x > \max(f)\}$ ($dom(f) = D_\alpha$ is finite).

Simple assignments of the form $y := t(\boldsymbol{z})$, where t is an elementary term, are evaluated in the usual way, f and Y are not changed.

An assignment of the form $y := f(z)$ is treated as follows. If $z \in dom(f)$, then return the corresponding value of f, do not change f or Y. If $z \notin dom(f)$, then set $f(z) = \min(Y)$, add z to D_α and delete $\min(Y)$ from Y.

An assignment of the form $y := \mu i \leq z.\ R(i)$ is treated in accordance with the evaluation procedure described above (f and Y are changed as specified in that procedure).

Clearly, the program $P_\alpha(a)$ will be eventually evaluated, and f will be defined at no more than q new points of the interval $[0, a]$. Hence, there will be a point of that interval, where f is not yet defined. Define the value of f at such points to be any sequence of different elements of Y, each of which is greater than the output of the program $P_\alpha(a)$. (This is sufficient for our purposes, for $\max_{i \leq a} f(i) < \bar{f}(a)$.) End of stage α of the construction of f.

Now we have to show that f as constructed really falsifies the program P_α on $[0, a]$. It suffices to show that the computation of P_α with the oracle function f yields the same result as our evaluation procedure. We prove this by induction on the length of P_α. Simple assignments obviously yield the same results.

Consider the crucial case of a μ-operator. Let f_1 and Y_1 be constructed by the evaluation procedure for this μ-operator. Obviously, f restricted to $[0, a]$ is an injective function extending f_1. Besides, by the construction only the values from Y_1 are added to f_1 at later stages of the evaluation procedure for the given μ-program. Therefore, Lemma 6 can be applied, which completes the induction step and the proof of the lemma.

Obviously, the function $\bar{f}$ belongs to $[\mathbf{C}(f), \mathbf{BR}]$ (see [10]). (It follows that the class $[\mathbf{C}(f), \mathbf{BR}]$ coincides with the elementary closure of f.) On the other hand, by Lemma 7 there is an f such that $\bar{f}$ is not in $[\mathbf{C}(f), \mathbf{M}]$. This completes the proof of Theorem 1.

Theorem 1 contrasts with the fact that doubly nested μ-operator suffices to generate the elementary closure.

Lemma 8. *For any function f,*

$$[[\mathbf{C}(f), \mathbf{M}], \mathbf{M}] = [\mathbf{C}(f), \mathbf{BR}] = \mathbf{E}(f).$$

Proof. If $R \in \mathbf{C}(f)$, then the predicate $\forall i \leq x\ R(i)$ belongs to $[\mathbf{C}(f), \mathbf{M}]$, for

$$\forall i \leq x\ R(i) \leftrightarrow x + 1 = \mu i \leq x. \neg R(i).$$

Defining $t(x) = \mu z \leq x.\ \forall u \leq x\ f(u) \leq f(z)$ yields

$$\max_{i \leq x} f(i) = f(t(x)) \in [[\mathbf{C}(f), \mathbf{M}], \mathbf{M}].$$

On the other hand, the relation $\bar{f}(x) \simeq y$ (see (1)) belongs to $[\mathbf{C}(f), \mathbf{M}]$, and there is an elementary function $g(x, y)$ such that for all x,

$$\bar{f}(x) = \mu u \leq g(x, \max_{i \leq x} f(i)). \bar{f}(x) \simeq u.$$

Hence, $\bar{f} \in [[\mathbf{C}(f), \mathbf{M}], \mathbf{M}]$ as a composition of two functions from this class. But $\mathbf{C}(\bar{f}) = \mathbf{E}(f)$.

In order to prove Theorem 2 we need a somewhat sharper separation result. The sequence of terms $t_0(x)$, $t_1(x,u_0)$, ... , $t_n(x,u_0,\ldots,u_{n-1})$ that appears in Lemma 3 and Corollary 1 can be considered as a kind of teacher-student game [7]: the student first tries to compute $\bar{f}(x)$ with a program $t_0(x)$. If the answer is correct, the student wins. If the answer is incorrect, the teacher has to show him/her a counterexample u_0 demonstrating that t_0 fails. The student then has the right to come up with a better solution $t_1(x,u_0)$ that may depend on the teacher's example u_0, and so on. The game has boundedly many rounds.

We have to show that the student who is only able to compute bounded query μ-programs cannot, in general, interactively compute $\bar{f}$ in the above sense. We will essentially use the fact that the teacher can only help the student boundedly many times. A sequence P of (reduced) μ-programs $P_0(x)$, $P_1(x,u_0)$, ... , $P_n(x,u_0,\ldots,u_{n-1})$ will be called an *interactive μ-program.*

Lemma 9. *There is a function $f:\mathbb{N}\to\mathbb{N}$ such that for any interactive μ-program P there is a number a and a sequence $u_0,\ldots,u_n$ of elements of $[0,a]$ such that the student looses the game $P(a)$ with the teacher's counterexamples $u_0,\ldots,u_n$:*

$$\begin{aligned}&f(u_0)\neq(P_0(a))_{u_0}\quad\wedge\\&f(u_1)\neq(P_1(a,u_0))_{u_1}\quad\wedge\quad\cdots\\&f(u_n)\neq(P_n(a,u_0,\ldots,u_{n-1}))_{u_n}.\end{aligned}$$

Proof. The argument is very similar to the one for Lemma 7. We only have to take care that the interval on which the game is being played has a place for all the teacher's answers.

Enumerate all possible interactive μ-programs. Let P_α be the program to be falsified at stage α, and let f be already defined on a finite domain D_α. Let q be greater than the total number of occurrences of the symbol f in P plus n. Fix an a such that $[0,a]$ has more than q elements outside D_α.

Evaluate P_0 as described in Lemma 7. Find the minimal u_0 such that $u_0\notin dom(f)$, define $f(u_0)$ to be the minimal element of Y bigger than the output of P_0 and throw out all elements of Y smaller than $f(u_0)$. Proceed in the same way with the evaluation of $P_1(a,u_0)$.

Obviously, after P_α is evaluated, less than q many new elements are added to the domain of f. Hence, all $u_0,\ldots,u_n$ are successively defined and falsify P_α.

This completes the proof of Theorem 2.

5 Further remarks

In this section we consider the strength of the induction and the least element principles for the classes of formulas between $\mathbf{C}(f)$ and $\Delta_0(f)$. First, we state some equivalences. Deductive equivalence of theories will be denoted $\equiv$.

Lemma 10. $I\Sigma^b_n(f)\equiv I\Pi^b_n(f)$.

Proof. This is proved by the usual trick of Parsons: induction for a formula $\varphi(x)$ can be reduced to the induction for $\neg\varphi(a \dot{-} x)$, where a is a free parameter. The reduction can be carried out in $\mathsf{EA}(f)$.

Lemma 11. $I\Pi^b_n(f) \equiv L\Sigma^b_n(f)$.

Proof. This follows by the usual proof of the least element principle by induction: $I_{\forall u \leq x\, \neg R(u)}$ implies L_R, and the class of $\Pi^b_n(f)$-formulas is closed under bounded universal quantifiers.

Notice that the dual form of this lemma for $n = 1$ is wrong. This can be seen from the next lemma.

Lemma 12. *The following theories are equivalent:*

1. $L\Sigma^b_1(f) \equiv L\mathbf{C}(f)$;
2. $L\Sigma^b_{n+1}(f) \equiv L\Pi^b_n(f)$, *for* $n \geq 1$.

Proof. We only prove Part 1. The proof of Part 2 is similar.

We are to derive inside $L\mathbf{C}(f)$ the formula L_φ, where $\varphi(a)$ has the form $\exists u \leq t(a)\ \psi(u, a)$ (possibly with some additional parameters). Here $t(a)$ is an elementary term.

Consider the set of pairs $S = \{\langle x, u\rangle \mid u \leq t(x)\}$ and order it as follows:

$$\langle x_1, u_1\rangle \prec \langle x_2, u_2\rangle \iff (x_1 < x_2 \vee (x_1 = x_2 \wedge u_1 < u_2)).$$

Obviously, there is an elementary isomorphism between $(S, \prec)$ and $(\mathbb{N}, <)$, so within $L\mathbf{C}(f)$ one can prove the least element principle for open formulas for $(S, \prec)$.

In order to prove the formula L_φ we formalize the following argument within $L\mathbf{C}(f)$. Assume $\exists u \leq t(a)\ \psi(u, a)$, then for some u satisfying $\psi(u, a)$ the pair $\langle a, u\rangle$ is in S. Applying the least element principle for $(S, \prec)$ to the $\mathbf{C}(f)$-formula $\psi_0(z) = \psi((z)_1, (z)_0)$ we obtain a pair $\langle x, v\rangle \preceq \langle a, u\rangle$ such that

$$\psi(v, x) \wedge \forall y, w\ (\langle y, w\rangle \prec \langle x, v\rangle \to \neg\psi(w, y)).$$

We claim that x is as required. Obviously, $x \leq a$ and $\exists v \leq x\ \psi(v, x)$. On the other hand, for all $y < x,\ w \leq t(y)$ there holds $\neg\psi(w, y)$, because all such pairs $\langle y, w\rangle$ belong to S and preceed $\langle x, v\rangle$ in the sense of $\prec$.

Corollary 3. *The following theories are equivalent:*

1. $L\mathbf{C}(f) \equiv L\Sigma^b_1(f) \equiv I\Sigma^b_1(f) \equiv I\Pi^b_1(f)$,
2. $L\Pi^b_1(f) \equiv L\Sigma^b_2(f) \equiv I\Sigma^b_2(f) \equiv I\Pi^b_2(f)$.

Let $\mathsf{EA}(\bar{f})$ denote the theory formulated in the language of EA with an extra function symbol $\bar{f}$. In addition to the axioms of EA it has the following two axioms:

1. $\forall x\ (\bar{f}(x) \in Seq \wedge lh(\bar{f}(x)) = x + 1)$,

2. $\forall x, y\, (x \leq y \to \bar{f}(x) = (\bar{f}(y) \upharpoonright x))$.

Here $(z \upharpoonright x)$ denotes the natural elementary function selecting the initial segment of length $x+1$ of a sequence coded by z. Further, let $I\mathbf{C}(\bar{f})$ denote the extension of $\mathsf{EA}(\bar{f})$ by the schema of induction for open formulas.

The following lemma is established in [1]. Its proof can be obtained by formalization of the equivalence $\mathbf{C}(\bar{f}) = \mathbf{E}(f)$.

Lemma 13. *$I\mathbf{C}(\bar{f})$ contains $L\Delta_0(\bar{f})$ and $L\Delta_0(f)$, under the interpretation of $f(x)$ as the term $(\bar{f}(x))_x$.*

Notice that bounding terms occurring in the instances of $L\Delta_0(\bar{f})$ may actually involve $\bar{f}$.

As it is shown in [1], $L\Delta_0(f)$ also naturally interprets $I\mathbf{C}(\bar{f})$, see equation (1). The following lemma is an improvement of this.

Lemma 14. *Under the natural interpretation of $\bar{f}$, $L\Pi_1^b(f)$ contains $L\mathbf{C}(\bar{f})$.*

Proof. We first recall that by the proof of Lemma 1

$$I\Sigma_2^b(f) \vdash \forall x \exists y \bar{f}(x) \simeq y.$$

Hence, the same formula is provable in $L\Pi_1^b(f)$.

Second, we use an argument similar to the proof of Proposition 5.11 from [1] (which, in turn, derives from [5]). $L\mathbf{C}(\bar{f})$ can be reduced to the least element principle for positive Σ_1^b-formulas in the graph of $\bar{f}$, more precisely, the formulas built up from elementary ones and $\bar{f}(x) = y$ using $\wedge$ and bounded existential quantifiers. This relies on the monotonicity of $\bar{f}$. The natural translation of the formula $\bar{f}(x) = y$ belongs to $\Pi_1^b(f)$, hence $L\mathbf{C}(\bar{f})$ is interpretable in $L\Sigma_2^b(f)$, which by Corollary 3 is equivalent to $L\Pi_1^b(f)$.

Thus, the hierarchy of least element schemata in our setting collapses above the level of $\Pi_1^b(f)$.

Corollary 4. $L\Pi_1^b(f) \equiv L\Delta_0(f)$.

Conclusion: The theories considered so far fall into four distinct classes:

$$\mathsf{EA}(f) \subset I\mathbf{C}(f) \subset L\mathbf{C}(f) \subset L\Pi_1^b(f) \equiv L\Delta_0(f).$$

Main theorem of this paper shows that $L\mathbf{C}(f)$ and $L\Delta_0(f)$ are really distinct. It is also possible to separate $I\mathbf{C}(f)$ from $L\mathbf{C}(f)$ and from $\mathsf{EA}(f)$. These separation results are tightly related to the question of positioning the schema of arithmetical Δ_1-induction in the hierarchy of subsystems of Peano arithmetic. The proofs of the latter two results will be given in a subsequent paper.

References

1. L.D. Beklemishev. Induction rules, reflection principles, and provably recursive functions. *Annals of Pure and Applied Logic*, 85:193–242, 1997.
2. L.D. Beklemishev. A proof-theoretic analysis of collection. *Archive for Mathematical Logic*, 37:275–296, 1998.
3. S.R. Buss. Introduction to Proof Theory. In S.R. Buss, editor, *Handbook of Proof Theory*, pages 1–78. Elsevier, North-Holland, Amsterdam, 1998.
4. P. Clote and J. Krajíček. Open problems. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory, and Computational Complexity*, pages 1–19. Oxford University Press, Oxford, 1993.
5. H. Gaifman and C. Dimitracopoulos. Fragments of Peano's arithmetic and the MDRP theorem. In *Logic and algorithmic (Zurich, 1980), (Monograph. Enseign. Math., 30)*, pages 187–206. Genève, University of Genève, 1982.
6. P. Hájek and P. Pudlák. *Metamathematics of First Order Arithmetic*. Springer-Verlag, Berlin, Heidelberg, New-York, 1993.
7. J. Krajíček. *Bounded arithmetic, Propositional logic, and Complexity theory*. Cambridge University Press, Cambridge, 1995.
8. L. Kristiansen. Fragments of Peano arithmetic and subrecursive degrees. Manuscript, 1998.
9. C. Parsons. Hierarchies of primitive recursive functions. *Zeitschrift f. math. Logik und Grundlagen d. Math.*, 14(4):357–376, 1968.
10. H.E. Rose. *Subrecursion: Functions and Hierarchies*. Clarendon Press, Oxford, 1984.