Subrecursion as Basis for a Feasible Programming Language

Paul J. Voda
Institute of Informatics, University of Bratislava
(voda@fmph.uniba.sk)

July 1994

Abstract

We are motivated by finding a good basis for the semantics of programming languages and investigate small classes in subrecursive hierarchies of functions. We do this with the help of pairing functions because in this way we can explore the amazing coding powers of S-expressions of LISP within the domain of natural numbers. In the process of doing this we introduce a missing stage in Grzegorczyk-based hierarchies which solves the longstanding open problem of what is the precise relation between the small recursive classes and those of complexity theory.

1 Introduction

We investigate subrecursive hierarchies based on pairing functions and solve a longstanding open problem in small recursive classes of what is the relationship between these and computational complexity classes (see [11]). The problem is solved by discovering that there is a missing stage in Grzegorczyk-based hierarchies [7, 11].

The motivation for this research comes from our search for a good programming language. We restrict our attention to declarative programming where we construct computable functions over some inductively presented domain. The domain of LISP, i.e. S-expressions, is an example of a simple, yet amazingly powerful, domain specified as words. We have designed and implemented two practical declarative programming languages Trilogy I and Trilogy II based on S-expressions with a single atom 0 (*Nil*) [13, 1]. This domain has been also investigated with the help of an imperative language for complexity purposes for instance in [5], or as a meta-language for the study of formal systems for constructive mathematics [4]. Since the domain of S-expressions with a single atom is denumerable it seems natural to identify it with the set of natural numbers. Functions of our programming language will become recursive functions. The identification is obtained by means of a suitable pairing function. Quite a few people have investigated properties of S-expressions but to our knowledge nobody has done it in the context of subrecursion. Yet, a feasible programming language should restrict itself to functions computable by binary coded Turing machines in polynomial time. This class is a subclass of elementary functions which is a small subclass of primitive recursive functions. Hence it seems natural to study the connection between a pairing-function-based presentation of primitive recursive function hierarchies with the usual presentation based on the successor function $s(x) = x + 1$. The relation to Grzegorczyk-based hierarchies should be central.

A computer scientist will note that the development of primitive recursive functions in the section 3 where we will operate on data structures such as lists and stacks will be just a declarative programming in the style of pure LISP. This should be contrasted with the usual development. The reader should not jump to the conclusion that since the development in the section 3 will be so straightforward it is trivial. Due to the brilliant insight of McCarthy in his design of LISP the development is practically bound to be simple. We order the presentation in the section 3 in such a way that we can quickly develop computational complexity classes in the sections 4 and 5. In contrast to the section 3, the development in the two sections will be rather complicated due mainly to the laborious process of estimating the growth rates of functions. The development will critically depend on the right choice of a pairing function.

Our contributions are **(i)** in the design of a clausal language for the definition of recursive functions essentially as a usable computer programming language (section 3), **(ii)** in the insight that the natural measure of S-expressions, i.e. the number of pairing operators (*cons*), should be tied to the size of natural numbers via our pairing function P. This gives us a characterization of pair-based function hierarchies by means of both Grzegorczyk-based hierachies (section 4) and Turing machines (section 5), **(iii)** but not before we find out that the jump operator of the Grzegorczyk hierachy, i.e. the operator of recursion, is too strong as to jump over an intermediate stage 2.5 of Grzegorczyk-based hierachies. The missing stage has a very pleasant Turing machine characterization in terms of polynomial time/space which was made possible **(iv)** by a weaker jump operator of pair iteration with logarithmically shorter length of iteration than recursion. This yields **(v)** recursion-theoretic closure conditions under which $P = NP$, $P = PSPACE$, and $PH = PSPACE$.

2 Pairing functions

All functions and predicates in this paper are total over the domain of natural numbers $\mathbf{N}$. It is well known that in the presence of a pairing function we can restrict our attention to unary functions and predicates. Unless we explicitly mention the arity of our functions and predicates they will be understood to be unary.

A binary function $\langle \cdot, \cdot \rangle$ is a *semi-suitable pairing function* if it is **(P1)**: a bijection from $\mathbf{N}^2$ onto $\mathbf{N} \setminus \{0\}$, and we have **(P2)**: $\langle x, y \rangle > x$ and $\langle x, y \rangle > y$. The condition **(P1)** assures the *pairing property* that from $\langle a, b \rangle = \langle c, d \rangle$ we get $a = c$ and $b = d$ and the property that 0 is the only *atom*, i.e. the only number not of the form $\langle x, y \rangle$.

Theorem 1 (Pair Induction) If $\langle \cdot, \cdot \rangle$ is a semi-suitable pairing function and R a predicate such that $R(0)$ and $\forall x \forall y (R(x) \land R(y) \rightarrow R\langle x, y \rangle)$ then $\forall x\, R(x)$.

Proof: By complete induction in a straightforward way ■

Theorem 2 (Pair Representation) If $\langle\cdot,\cdot\rangle$ is a semi-suitable pairing function then every natural number has a unique *pair representation* as a term obtained from 0 by finitely many applications of the pairing function.

Proof: By pair induction taking $R(x)$ to hold if x *has a unique representation.* ■

We will abbreviate $\langle x,\langle y,z\rangle\rangle$ to $\langle x,y,z\rangle$ and when discussing only unary functions we will write x,y for $\langle x,y\rangle$. Thus '$\cdot,\cdot$' can be viewed as an infix pairing operator with a lowest precedence where, for instance, $x+y,z$ stands for $(x+y),z$.

Theorems 1 and 2 guarantee that for a semi-suitable pairing function **(a):** every number x is either 0 or it can be uniquelly written in the form $x_1,x_2,\ldots,x_n,0$ for some $n\geq 1$ and numbers x_i. Thus every number codes a single finite sequence over numbers (codes of finite sequences are called *lists* in the computer science), and vice versa. **(b):** There exist unique *pair size* $|x|$ and *length* $Len(x)$ functions such that $|0|=0$, $|x,y|=|x|+|y|+1$, and $Len(0)=0$ and $Len(x,y)=Len(y)+1$. The function $Len(x)$ gives the length of the finite sequence coded by x. We have $Len(x)\leq|x|$.

There are many semi-suitable pairing functions. For instance, if we offset the standard recursion-theoretic pairing function J [3] by one, i.e. if $J(x,y)=((x+y)\cdot(x+y+1))\div 2+x+1$ we get a semi-suitable pairing function. The function J is not good for our purposes as it is not a *suitable pairing function* satisfying the additional condition **(P3)**: $|x|=\Theta(\log(x))$. We will see in the section 5 that a suitable pairing function gives a rise to the classes of functions with very desirable properties of computational complexity.

Let us temporarily assume that there is a binary function P such that the following sequence enumerates all natural numbers in the increasing order:

$$0|0,0|0,0,0|(0,0),0|0,0,0,0|0,(0,0),0|(0,0),(0,0)|(0,0,0),0|((0,0),0),0|\cdots \tag{1}$$

The sequence is obtained by ordering the pair representation terms on pair size and within the same pair size lexicographically. Clearly, P (if it exists) is a uniquely determined semi-suitable pairing function. The closed interval

$$[0,0,\cdots,0,0\ ;\ (\cdots((0,0),0),\cdots,0),0] \tag{2}$$

where both bounds have the pair size $n\geq 1$ contains all numbers with the pair size n. The lower bound of the interval is a *right-leaning number* while the upper bound is a *left-leaning number*. The length of the interval is the number of ways an expression consisting of n binary infix operators can be parenthesized. These numbers are known as *Catalan numbers* $C(n)=\frac{1}{n+1}\cdot\binom{2n}{n}$ (see [6]). By convention $C(0)=1$ and this is also the length of the closed interval $[0;0]$ consisting of the numbers of the pair size 0. By a straightforward manipulation of binomial coefficients we get $C(n+1)=\frac{4n+2}{n+2}\cdot C(n)$. From this by a simple induction proof we get

$$2^{n-1}\leq C(n)\leq 4^n. \tag{3}$$

The function

$$\sigma(n)=\sum_{i<n}C(i) \tag{4}$$

yields the minimal number with the pair size n. For all n the numbers in the closed interval $[\sigma(n); \sigma(n+1)-1]$ are exactly the numbers with the pair size n. We clearly have

$$|x| = \mu\, n \leq x\,[\, x < \sigma(n+1)\,] \tag{5}$$

where μ is the operator of *bounded minimum* (see [11]).

For $n \geq 1$ the interval (2) can be partitioned into a sequence of consecutive intervals $\mathcal{I}_i = [i, \sigma(n-1-|i|)\ ;\ i, \sigma(n-|i|)-1]$ where $0 \leq i < \sigma(n)$. The length of $\mathcal{I}_i$ is $C(n-1-|i|)$. Hence for $x < \sigma(n)$

$$x, \sigma(n-1-|x|) = \sigma(n) + \sum_{i<x} C(n-1-|i|). \tag{6}$$

The function $Ro(x) = x - \sigma\,|x|$ gives the offset of the number x from the least number of the same pair size. If for two numbers x and y we set $n = |x|+|y|+1$ we can see that $x, y = (x, \sigma\,|y|) + Ro(y)$ because the number x, y occurs at the offset $Ro(y)$ in the interval $\mathcal{I}_x$ with the lower bound $x, \sigma\,|y|$. Using (6) we get

$$P(x,y) = \sigma(|x|+|y|+1) + Ro(y) + \sum_{i<x} C(|x|+|y|-|i|). \tag{7}$$

We can conclude that if the function P exists then (7) must hold. Vice versa, when we define P by (7) where $|x|$ is defined by (5) we can see that the sequence (1) consists of all natural numbers in the increasing order. So the function P does exists and it is a semi-suitable pairing function strictly monotone in both arguments. We now show that P satisfies also the condition **(P3)**. We clearly have $C(n) \leq \sigma(n+1)$, so for $|x| \geq 2$ we get $2^{|x|-2} \leq C(|x|-1) \leq \sigma\,|x| \leq x$. Hence for all x

$$2^{|x|} \leq 4 \cdot x + 1. \tag{8}$$

Each of the intervals $\mathcal{I}_0$ through $\mathcal{I}_{\sigma(n)-1}$ is non-empty and so $\sigma(n) \leq C(n)$ which holds also for $n = 0$. Thus

$$x < \sigma(|x|+1) \leq C(|x|+1) \leq 4^{|x|+1} = 2^{2\cdot|x|+2}. \tag{9}$$

Let us denote by $d(x)$ the *binary size* function yielding the number of bits in the binary representation of x ($d(0) = 0$). We clearly have $d(x) = \Theta(\log(x))$. From (8) we get $|x|+1 = d(2^{|x|}) \leq d(4 \cdot x + 1) = d(x) + 2$, i.e. $|x| \leq d(x) + 1$. From (9) we get $d(x) \leq d(2^{2\cdot|x|+2}) = 2 \cdot |x| + 3$. Thus the the condition **(P3)** holds and we have:

Theorem 3 The function P defined by (7) is a suitable pairing function. ■

From now on $\langle x, y\rangle$ and x, y are abbreviations for $P(x, y)$.

3 Pair-Based Functions and Classes

The development of function classes discussed in this section does not require the property **(P3)**. Any primitive recursive semi-suitable pairing function p can be used instead of P provided we are able to derive the successor function and unary iteration in a way similar to the derivation of arithmetic below.

The *identity* function is $I(x) = x$. The *zero* function is $Z(x) = 0$. The *first (H) and second (T) projection* functions are such that $H(0) = T(0) =$

0, $H(x,y) = x$, and $T(x,y) = y$. The *conditional* function D is such that $D(0,x,y) = y$, $D((v,w),x,y) = x$, and $D(0) = D(x,0) = 0$. The function f is obtained from the functions g and h by *(unary) composition* if $f(x) = g\,h(x)$ and by *pairing* if $f(x) = g(x), h(x)$. Functions *simple (in a class of functions $\mathcal{F}$)* are generated from $(\mathcal{F})$, I, Z, and D by composition and pairing.

Theorem 4 A class of functions simple in $\mathcal{F}$ is closed under *explicit definitions* of the form $f(x) = \boldsymbol{a}$ where $\boldsymbol{a}$ is a term constructed from the variable x and numerals by pairing $(\boldsymbol{b}, \boldsymbol{c})$ and applications $g(\boldsymbol{b})$ where the function g is a function simple in $\mathcal{F}$.

Proof: By a straightforward induction on the structure of $\boldsymbol{a}$. We may assume that all numerals in $\boldsymbol{a}$ are in the pair representation. ■

The projection functions are simple as they have explicit definitions $H(x) = D(1,x)$ and $T(x) = D(0,x)$. An example of a function f simple in g and h is

$$f(x) = D(x, D(H(x), 0, D(g\,T(x), h(T\,g\,T(x), H\,g\,T(x)), 0)), c). \tag{10}$$

Definitions like (10) are made readable by a *clausal language* where we define the same f by an *explicit clausal definition* as

$$\begin{aligned}
&f((v,w),x) = 0\\
&f(0,x) = h(w,v) \leftarrow g(x) = v, w\\
&f(0,x) = 0 \leftarrow g(x) = 0\\
&f(0) = c.
\end{aligned}$$

Th clauses, when read as inverted implications, express properties of the defined function. The first and third clauses can be omitted as the defined function then obtains the value 0 by *default* when no clause can be satisfied for a given argument. Clauses can be given in any order (we have listed the clauses for f in the order obtained by removing the conditionals from the definition (10)). Clauses must be presented in such a way that the conversion from a clausal definition to its initial form should be always possible.

We do not have the space to dwell on the details of our clausal language and hope that the clausal definitions given in this paper will be simple enough so the reader can reconstruct the initial form of the definition. In the case of explicit definitions the initial form is always $f(x) = \boldsymbol{a}$. We permit the clausal language to be used with inductive definitions where the initial form will be a schema of inductive definition. The development of the clausal language should be viewed from the perspective of its eventual implementation on computers. We offer here no details on how one can go about the automatic compilation.

We now present three schemas of pair-based inductive definitions which can be used as initial forms of clausal definitions. The function f is defined by the *unary iteration* of g if $f(0) = 0$, $f(0,y) = y$, and $f(s(x),y) = g\,f(x,y)$. We write $g^x(y)$ as an abbreviation for the application $f(x,y)$. The function f is defined by the *pair iteration* of g if $f(0) = 0$ and $f(x,y) = g^{|x|}(y)$. The function f is defined by *pair recursion* from g and h if $f(0) = 0$, $f(0,y) = g(y)$, and $f((v,w),y) = h(v,w,y,f(v,y),f(w,y))$.

The classes $\mathcal{PR}_1$, $\mathcal{PR}_2$, and $\mathcal{PR}_3$ are generated from I, H, and T by composition pairing and respectively by pair recursion, pair iteration, and unary iteration. We call the functions of $\mathcal{PR}_1$ *primitive pair recursive* functions.

Theorem 5 The classes $\mathcal{PR}_i$ are closed under explicit clausal definitions.

Proof: Clearly, it is sufficient to show that the classes $\mathcal{PR}_i$ are simple in themselves, i.e. that each of them contains the functions Z and D. The function Z is derived in $\mathcal{PR}_i$ by the iteration of T. In the class $\mathcal{PR}_1$ this is done by first defining f_1 by pair recursion from I and $h(x) = T\,T\,T\,T\,T(x)$ where h is obtained by a repeated composition. Note that we have $f_1((v,w),y) = T\,f_1(w,y)$. Then we define $f_2(x) = I(x), I(x)$ by pairing, and finally $Z(x) = f_1\,f_2(x)$ by composition. The derivations of Z in $\mathcal{PR}_2$ and $\mathcal{PR}_3$ are similar but we define f_1 by pair iteration and unary iteration of T respectively.

The derivation of D in $\mathcal{PR}_1$ is by pair recursion from T and $h(x) = H\,H\,T\,T(x)$. Note that we then have $D(0,y) = T(y)$ and $D((v,w),y) = H(y)$. The derivation of $D \in \mathcal{PR}_2$ is by composition $D(x) = T\,f(x)$ where f is defined by pair iteration of $g(x) = H(x), H(x)$ obtained by pairing. If we define f by unary iteration of g we get $D \in \mathcal{PR}_3$. ■

We can now use the clausal language with any of the three inductive schemas. The functions $\oplus$, $\otimes$, $\uparrow$, written in infix notation e.g. $x \oplus y$ instead of $\oplus(x,y)$, are obtained in $\mathcal{PR}_1$ by pair recursion:

$$
\begin{aligned}
&0 \oplus y = y\\
&(v,w) \oplus y = v, w \oplus y\\
&Rt(0) = 0\\
&Rt(v,w) = 0, Rt(v) \oplus Rt(w)\\
&x \otimes y = f_1(Rt(x), y)\\
&f_1(0,y) = 0\\
&f_1((v,w),y) = y \oplus f_1(w,y)\\
&x \uparrow y = f_2(Rt(y), x)\\
&f_2(0,x) = 1\\
&f_2((v,w),x) = x \otimes f_2(w,x).
\end{aligned}
$$

The function $\oplus$ is the *list concatenation* function. The function Rt is obtained by *parameterless pair recursion* for which $\mathcal{PR}_1$ is easily seen closed. $Rt(x)$ yields a right-leaning number of the same pair size as x: $Rt(x) = \sigma\,|x|$. We have $|Rt(x)| = Len\;Rt(x) = |x|$. By pair induction we can derive $|x \oplus y| = |x| + |y|$, $|x \otimes y| = |x| \cdot |y|$, and $|x \uparrow y| = |x|^{|y|}$.

The function $Lt(x) = \sigma(|x|+1) - 1$ yielding a left left-leaning number of the same pair size is obtained as primitive pair recursive by $Lt(x) = f\;Rt(x)$ where $f(0) = 0$ and $f(v,w) = f(w), 0$.

We identify a predicate R with its *characteristic function* $R_*(x) = y \leftrightarrow R(x) \land y = 1 \lor \neg R(x) \land y = 0$. For a function class $\mathcal{F}$ we denote by $\mathcal{F}_*$ the class of $0,1$-valued functions of $\mathcal{F}$. We say that the predicate R *is from* $\mathcal{F}$, and sometimes write $R \in \mathcal{F}$, if $R_* \in \mathcal{F}_*$. The predicate $Left(x)$ true of left-leaning numbers is defined primitive pair recursive by a clausal definition

$$
\begin{aligned}
&Left(0)\\
&Left(v,w) \leftarrow w = 0 \land Left(v).
\end{aligned}
$$

Clausal definitions of predicates should be viewed as abbreviations for clausal definitions of their characteristic functions. If we replace $Left(x)$ by $Left_*(x) = 1$ above we get such a definition. Note that the clauses for $\neg Left(x)$, i.e. for $Left_*(x) = 0$, are obtained by default. We can similarly define the primitive pair recursive predicate *Right* holding of right-leaning numbers.

We now turn to the arithmetic functions. The successor and predecessor functions are obtained by the case analysis of the enumeration of pairs (1). We derive the successor function in $\mathcal{PR}_1$ by a parameterless pair recursion:

$$
\begin{aligned}
&s(0) = 1\\
&s(v,w) = v, s(w) \leftarrow \neg \mathit{Left}(w)\\
&s(v,w) = s(v), \mathit{Rt}(w) \leftarrow \mathit{Left}(w) \wedge \neg \mathit{Left}(v)\\
&s(v,w) = 0, \mathit{Rt}(v,w) \leftarrow \mathit{Left}(w) \wedge \mathit{Left}(v) \wedge w = 0\\
&s(v,w) = \mathit{Rt}(0,v), \mathit{Rt}(w_1) \leftarrow \mathit{Left}(w) \wedge \mathit{Left}(v) \wedge w = w_1, w_2.
\end{aligned}
$$

The *predecessor* function $p(0) = 0$ and $p\,s(x) = x$ is derived as primitive pair recursive in a similar way with with the help of Lt and Right. Before we can get additional arithmetic functions we need $\mathcal{PR}_1$ closed under unary iteration:

Theorem 6 The classes $\mathcal{PR}_i$ consist exactly of primitive pair recursive functions.

Proof: $\mathcal{PR}_1 \subset \mathcal{PR}_2$: It is clearly sufficient to show $f \in \mathcal{PR}_2$ for f derived by the pair recursion from g and h in $\mathcal{PR}_2$. Derive f in $\mathcal{PR}_2$ by pair iteration:

$$
\begin{aligned}
&f(x,y) = v \leftarrow h_1^{|x,x,x|}(y, 0, (0,x), 0) = y_1, ((v, x_1), s), i\\
&h_1(y, s, (0,0), i) = y, ((g(y), 0), s), i\\
&h_1(y, s, (0, v, w), i) = y, s, (0, w), (0, v), 0, i\\
&h_1(y, ((fv, v), (fw, w), s), 0, i) = y, ((h(v, w, y, fv, fw), v, w), s), i\\
&h_1(y, s, 0) = y, s, 0.
\end{aligned}
$$

The 'second' argument s of h_1 is an output stack where the function values $f(z,y)$ of components z of x are assembled in the form $f(z,y), z$. The 'third' argument i is an input stack for breaking down the components z of x. The components are stored in the form $(0,z)$. The marker 0 is pushed on the input stack in the second clause for h_1 to indicate that when it eventually appears at the top of the stack in the third clause for h_1 the values $f(v,y), v$ and $f(w,y), w$ will be on the top of the output stack. The needed length of iteration of h_1 is $3 \cdot |x| + 1 < |x,x,x|$ as we need two iterations for each comma and one iteration for each 0 in the pair representation of x.

$\mathcal{PR}_2 \subset \mathcal{PR}_3$: It is sufficient to show $f \in \mathcal{PR}_3$ for f derived by pair iteration of $g \in \mathcal{PR}_3$. Derive f in $\mathcal{PR}_3$ by unary iteration:

$$
\begin{aligned}
&f(x,y) = a \leftarrow g_1^{x,x}(y, x, 0) = a, s\\
&g_1(a, 0, s) = a, s\\
&g_1(a, (v,w), s) = g(a), v, w, s\\
&g_1(a, 0) = a, 0.
\end{aligned}
$$

The iteration of g_1 goes long enough $2 \cdot |x| + 1 = |x,x| \leq x,x$ to empty the 'second' argument s of g_1 which acts as an input stack for breaking the original argument x into its components. The 'first' argument a is an accumulator where g is applied to each time we have a pair on the top of the stack.

$\mathcal{PR}_3 \subset \mathcal{PR}_1$: Assume f defined by the unary iteration of $g \in \mathcal{PR}_1$ and show $f \in \mathcal{PR}_1$. Derive f in $\mathcal{PR}_1$ by pair recursion:

$$
\begin{aligned}
&f(x,y) = y_1 \leftarrow f_1(\mathit{Rt}(9 \uparrow (0,x)), x, y) = x_1, y_1\\
&f_1(0, x, y) = x, y\\
&f_1((v,w), x, y) = 0, y_1 \leftarrow f_1(w, x, y) = 0, y_1\\
&f_1((v,w), x, y) = p(v_1, w_1), g(y_1) \leftarrow f_1(w, x, y) = (v_1, w_1), y_1.
\end{aligned}
$$

The auxiliary function f_1 accepts as a parameter and yields numbers of the form c, a where c is a counter of iterations and a an accumulator where g is repeatedly applied to. This works provided the length of recursion is at least x. By (9) we have

$$Len\ Rt(9 \uparrow (0,x)) = |9 \uparrow (0,x)| = |9|^{|0,x|} = 4^{|x|+1} > x. \ \blacksquare$$

An *unary equivalent* of an n-ary function f $(n \geq 2)$ is any function g such that $f(x_1, \cdots, x_n) = g\langle x_1, \cdots, x_n\rangle$. Unary equivalents of arithmetic functions $+$, $\dot{-}$ (modified subtraction), $\cdot$, $\div$ (integer division), and x^y can be derived in $\mathcal{PR}_i$ by unary iteration simulating the usual recursion. The function d is derived in $\mathcal{PR}_i$ by a repeated halving.

4 Pair-Based Hierarchies

We assume that the reader is familiar with the Grzegorczyk hierarchy $\mathcal{E}^i$ [7] which we will need from the *multiplicative* stage $\mathcal{E}^2$ onwards. Rose in [11] gives a good discussion of the topic. The *hierachy functions* generating the classes $\mathcal{E}^i$ are $E_2(x) = x^2 + 2$, and and $E_{i+3}(x) = E^x_{i+2}(2)$. The reader will note that in order to get $E_i \in \mathcal{E}^i$ we have increased the indices of Rose's functions by one. Functions E_i are strictly monotone, E_{i+1} *bounds* (*dominates*) E_i, i.e. $E_i(x) \leq E_{i+1}(x)$ for all x. We also have $E^x_i(y) \leq E_{i+1}(x+y)$. Proofs are in [11].

The *subexponential* stage 2.5 between the multiplicative and *elementary* (*exponential*) stage 3 is the announced missing stage in the Grzegorczyk hierachy. We *augment* the Grzegorczyk hierarchy by adding the class $\mathcal{E}^{2.5}$ generated in the usual way from the binary *subexponential* hierarchy function $E_{2.5}(x,y) = x^{d(y)}$. It is easy to see that all functions $f(x_1, \cdots, x_n)$ in $\mathcal{E}^{2.5}$ are bounded by $2^{p(d(x_1), \cdots, d(x_n))}$ for a polynomial $p(x_1, \cdots, x_n)$ and so the class $\mathcal{E}^{2.5}$ lies strictly between the classes $\mathcal{E}^2$ and $\mathcal{E}^3$.

Pair hierachy functions F_i generate our *pair-based hierachies.* We set $F_2 = I$, $F_{2.5} = \otimes$, $F_3(x) = g^{|x|}(2)$ where $g(x) = x \otimes x \oplus 2$, and $F_{i+4}(x) = F^{|x|}_{i+3}(2)$. We have $F_i \in \mathcal{PR}_j$. The functions F_{i+3} have been choosen in such a way that we have $|F_{i+3}(x)| = E_{i+3}|x|$. The proof is left to the reader.

A function operator yielding the function f is *limited* if f is bounded by a function j which is an argument of the limited operator. To every operator introduced sofar we clearly have a limited version by adding a new argument function j.

Definition 4.1 (Pair hierarchies) The classes $\mathcal{P}^i$ $(i = 2, 2.5, 3, 4, \cdots)$ of the *pair hierarchy* are generated from I, H, T, and F_i, by composition, pairing, and limited pair iteration.

The classes $\mathcal{PM}^i$ of the *pair bounded minimum hierarchy* are obtained by adding to the classes $\mathcal{P}^i$ the operator of *bounded unary minimum*: $f(0) = 0$, $f(y,x) = \mu\, z \leq y\,[\,R(z,x)\,]$ yielding the least $z \leq y$ to satisfy R or 0 if there is no such.

The classes $\mathcal{PO}^i$ of the *pair order hierarchy* are obtained by replacing in the classes $\mathcal{P}^i$ limited pair iteration by limited unary iteration.

We will write $\mathcal{F}^i$ when we mean a class at the i-th stage of any of the three pair hierarchies. We did not define the classes $\mathcal{F}^i$ of pair hierarchies below the multiplicative stage $\mathcal{F}^2$, i.e. for $i = 0$ and $i = 1$. As it is well known no pairing function can be introduced in these stages. We only note here that by limiting the pairing operator we can extend the pairing hierarchies to the stages zero or one. We now prove a number of lemmas leading to the theorems 20 and 22.

Lemma 7 The classes $\mathcal{F}^i$ are closed under explicit clausal definitions.

Proof: This follows from the proof of the theorem 5 for the classes $\mathcal{PR}_2$ and $\mathcal{PR}_3$ by observing that when deriving Z from f_1 by pair (unary) iteration we have $f_1(x) \leq T(x)$ and when deriving D from f we have $f(x) \leq T(x), T(x)$. ■

When a function f is obtained by a limited operator we have $f(x) \leq j(x)$ from which $|f(x)| \leq |j(x)|$ follows. Vice versa, the latter condition implies $f(x) < 0, j(x) = j_1(x)$ where the explicitly defined function j_1 is in the same class as j. As the two bounding conditions are equivalent we will be using both of them. We say that a function f is *monotone in the pair size* if $|x| \leq |y| \rightarrow |f(x)| \leq |f(y)|$. Note that the pairing function P is monotone in the pair size of both arguments.

Lemma 8 $\oplus \in \mathcal{F}^i$.

Proof: We derive by the pair iteration:

$$x \oplus y = Reva(Reva(x, 0), y)$$
$$Reva(x, y) = y_1 \leftarrow f^{|x|}(x, y) = x_1, y_1$$
$$f(0, y) = 0, y$$
$$f((v, w), y) = w, v, y.$$

A computer programmer will recognize that the function *Reva* is the so called accumulator version of the list reversal function *Rev*. We have $Reva(x, y) = Rev(x) \oplus y$, i.e. $Rev(x) = Reva(x, 0)$. The pair iteration of f is limited as we have $|f^{|n|}(x)| = |x| \leq |T(n, x)|$. It should be clear that we can replace the limited pair iteration by the limited unary iteration because $|x| \leq x$. ■

Lemma 9 For any polynomial $p(x_1, \cdots, x_n)$ we can define a function j such that $|j(x_1, \cdots, x_n)| = p(|x_1|, \cdots, |x_n|)$. If p is a linear form then $j \in \mathcal{F}^i$, otherwise j is in any class $\mathcal{F}^i$ containing $\otimes$.

Proof: By induction on the structure of p. If $p = c$ we set $j(x) = \sigma(c)$. If $p = x_i$ we define j by a clause $j(x_1, \cdots, x_n) = x_i$. If $p = p_1 + p_2$ we use the clausal definition $j(x_1, \cdots, x_n) = j_1(x_1, \cdots, x_n) \oplus j_2(x_1, \cdots x_n)$ where j_1 and j_2 are obtained from induction hypotheses. The case $p = p_1 \cdot p_2$ is similar where we use $\otimes$ instead of $\oplus$. We note that if in the case of multiplication one of the polynomials is a constant c we can define j by a $\sigma(c)$-fold composition of $\oplus$. If p is a linear form the function $\otimes$ is not needed and we have $j \in \mathcal{F}^i$. ■

Lemma 10 Let $f \in \mathcal{F}^i$. If $i \leq 2.5$ then there is a polynomial $p(n)$ such that $|f(x)| \leq p\,|x|$. If $i = 2$ then p is a linear form. If $i \geq 3$ then for some c we have $|f(x)| \leq |F_i^c(x)|$. Moreover, $|f(x)| \leq |j(x)|$ for a monotone in the pair size $j \in \mathcal{F}^i$.

Proof: The proof of the first part is by induction on the construction of f. The only interesting cases are those of composition and pairing. For $i \leq 2.5$ we have by induction hypotheses $|g(x)| \leq p\,|x|$ and $|h(x)| \leq q\,|x|$ and then

$$\begin{aligned} |f(x)| &= |g\,h(x)| \leq p\,|h(x)| \leq p\,q\,|x| \\ |f(x)| &= |g(x), h(x)| = |g(x)| + |h(x)| + 1 \leq p\,|x| + q\,|x| + 1. \end{aligned}$$

The polynomials $p\,q(n)$ and $p(n) + q(n) + 1$ are linear forms if both p and q are. For $i \geq 3$ we get similarly:

$$\begin{aligned} &|f(x)| = |g\,h(x)| \leq |F_i^m\,h(x)| = E_i^m\,|h(x)| \leq \\ &\quad E_i^m\,|F_i^n(x)| = E_i^m\,E_i^n\,|x| = E_i^{m+n}|x| = |F_i^{m+n}(x)|. \end{aligned}$$

The case of pairing is similar. For the second part of the theorem we use in the case $i \leq 2.5$ the function j obtained by the lemma 9 and in the case $i \geq 3$ we set $j(x) = F_i^c(x)$. From the monotonicity of $+$, $\cdot$ and E_i we can easily see that j is monotone in the pair size. ∎

Lemma 11 The classes $\mathcal{F}^i$ are closed under limited pair iteration. Hence $\mathcal{P}^i \subset \mathcal{PM}^i, \mathcal{PO}^i$ and $\mathcal{P}^2 \subset \mathcal{F}^i$.

Proof: The second claim trivially follows from the first one. The third claim follows from the first two as $F_2 = I \in \mathcal{F}^i$. The first claim is non-trivial only when $F^i = \mathcal{PO}^i$. For this case we use the derivation of f from the inclusion $\mathcal{PR}_2 \subset \mathcal{PR}_3$ of the theorem 6 and estimate the size of the iterated g_1 under the assumption $|g^{|x|}(y)| \leq |j(x,y)|$ where we can assume that j is monotone in the pair size. For some function k and a number s_1 we have $g_1^x(a,s) = g^{k(x,s)}(a), s_1 = g^{|\sigma\,k(x,s)|}(a), s_1$ where clearly $|s_1| \leq |s|$ and $|\sigma\,k(x,s)| = k(x,s) \leq |s|$. Hence we have

$$|g_1^x(a,s)| \leq |j(\sigma\,k(x,s),a)| + |s| + 1 \leq |j(s,a)| + |s| + 1 = |j(s,a), s|. \;\blacksquare$$

Lemma 12 If $2 \leq i < n$ then $F_i \in \mathcal{F}^n$.

Proof: By the lemma 11 it suffices to derive F_i in $\mathcal{P}^n$. Fix n, and consider the sequence $F_2,\ F_{2.5},\ F_3\ \ldots F_{n-1}$. Clearly $F_2 = I \in \mathcal{P}^n$. For $F_{2.5} = \otimes$ use the clausal definition $x \otimes y = T\,f^{|x|}(y,0)$ with $f(y,a) = y, y \oplus a$. We note that $|x|^2 \leq E_2|x| \leq E_n|x| = |F_n(x)|$. Hence

$$|f^{|x|}(y,a)| = |y,a| + |x| \cdot |y| \leq |y,a| + |x \oplus y|^2 \leq |(y,a) \oplus F_n(x \oplus y)|.$$

The remaining functions in the sequence are obtained by the pair iteration of preceding ones and by deriving a $\mathcal{P}^n$ bound on them in a similar way. ∎

Lemma 13 $\uparrow \in \mathcal{P}^3$.

Proof: We use the clausal definition $x \uparrow y = T\,f^{|y|}(x,1)$ with $f(x,a) = x, x \otimes a$. We have $|f^{|y|}(x,a)| = |x| + |a| \cdot |x|^{|y|} + 1$. We can bound the iteration of f in $\mathcal{P}^3$ if we find a $\mathcal{P}^3$ function b such that $|x|^{|y|} \leq |b(x,y)|$. We have

$$|x|^{|y|} \leq x^{|y|} \leq 2^{d(x)\cdot|y|} \leq 2^{(2\cdot|x|+3)\cdot|y|} = 2^{|(1,x,x)\otimes y|}. \tag{11}$$

By induction on x we easily prove $2^{2^x} \leq E_3(x)$. Hence

$$|x|^{|y|} \leq 2^{|(1,x,x)\otimes y|} \leq 2^{2^{|(1,x,x)\otimes y|}} \leq E_3|(1,x,x)\otimes y| = |F_3((1,x,x)\otimes y)|. \;\blacksquare$$

Lemma 14 $\mathcal{F}^2 \subset \mathcal{F}^{2.5} \subset \mathcal{F}^3 \subset \mathcal{F}^4 \subset \cdots$ and the inclusions are strict.

Proof: The inclusions hold by the lemma 12. For the strictness we use the lemma 10. We cannot have $|x \otimes x| = |x|^2 \leq p|x|$ for a linear form p, hence $\otimes \notin \mathcal{F}^2$. We cannot have $|x \uparrow x| = |x|^{|x|} \leq p|x|$ for a polynomial p, hence $\uparrow \notin \mathcal{F}^{2.5}$. Given a constant c, choose an x such that $|x| > c$. We have

$$|F_{i+3}^c(x)| = E_{i+3}^c|x| \leq E_{i+4}(|x| + c) < E_{i+4}(2 \cdot |x|) = |F_{i+4}(x \oplus x)|.$$

The $\mathcal{F}^{i+4}$ function $f(x) = F_{i+4}(x \oplus x)$ cannot be in $\mathcal{F}^{i+3}$ as it would exceed its own bound. ■

Bounding conditions of the form $|g^n(x)| \leq |j(x)|$ are not always easy to prove as we cannot assume anything about x. An assumption is needed when the iteration has to be *initialized*, for instance, by setting up the stacks. The following lemma is then useful.

Lemma 15 Given an iteration $f_1(x) = g^{|t(x)|}\, i(x)$ ($f_1(x) = g^{t(x)}\, i(x)$) such that $R(x) \rightarrow |g^n\, i(x)| \leq |b(x)|$ then there is a function f such that $R(x) \rightarrow f(x) = f_1(x)$. The function f is in the same class $\mathcal{F}^i$ ($\mathcal{F}^i$ must be $\mathcal{PO}^i$ for the unary iteration) as the functions t, g, i, and b.

Proof: We call the function g_1 a *bounded version* of g if

$$\begin{aligned} &g_1(b, x) = b, y \leftarrow g(x) = y \wedge |y| \leq |b| \\ &g_1(b, x) = b, x \leftarrow g(x) = y \wedge |y| \not\leq |b|. \end{aligned}$$

The pair size comparing predicate $|x| \leq |y|$ is derived by the pair iteration:

$$\begin{aligned} &|x| \leq |y| \leftarrow h^{|x,x,y,y|}((x,0), y, 0) = 0, s_2 \\ &h(((v,w), s_1), s_2) = (v, w, s_1), s_2 \\ &h((0, s_1), ((v,w), s_2) = (0, s_1), v, w, s_2 \\ &h((0, s_1), (0, s_2)) = s_1, s_2 \\ &h((0, s_1), 0) = (0, s_1), 0 \\ &h(0, s_2) = 0, s_2. \end{aligned}$$

We try to exhaust two stacks s_1 and s_2 containing the components of x and y respectively. The stack s_2 should not be exhausted before s_1. We do not need more than one iteration of h for each zero and comma in x and y. The predicate is in $\mathcal{P}^2$ as the pair size of h does not increase.

We define $f(x) = T\, g_1^{|t(x)|}(b(x), i(x))$ (we use unary iteration in the parenthesized case). We have $|g_1^n(b, y)| \leq |b, y \oplus b|$ and so $f \in \mathcal{F}^i$. Finally, by induction on n we get $R(x) \rightarrow g_1^n(b(x), i(x)) = b(x), g^n\, i(x)$ from which the desired property immediately follows. ■

Lemma 16 The classes $\mathcal{F}^i$ for $i \geq 2.5$ are closed under limited pair recursion. The classes $\mathcal{F}^2$ are closed under limited pair recursion when the bounding condition is independent of the parameter: $|f(x, y)| \leq |j(x)|$. This, clearly, includes limited parameterless pair recursion.

Proof: We use the inclusion $\mathcal{PR}_1 \subset \mathcal{PR}_2$ of the theorem 6 under the assumption $|f(x, y)| \leq |j(x, y)|$ where we can assume that j is monotone in the pair size. We estimate the increase in the pair size of $h_1^n(y, 0, (0, x), 0)$ for one iteration. The first clause of h_1 is used $|x| + 1$ times and the pair size increases by $|f(0,y)|$, the second clause is used $|x|$ times and the increase is 2, the third clause is used $|x|$ times and the increase is $|f((v,w),y)| - |f(v,y)| - |f(w,y)|$ where v, w is a component of x. We wish to find functions q and b such that $|h_1^n(y,0,(0,x),0)| \leq |y,0,(0,x),0| + q(x,y) = |b(x,y)|$ and apply the lemma 15 with the condition $R = \mathbf{N}$. For $i \geq 2.5$ it is sufficient to take $q(x,y) = |j(x,y)| \cdot (|x|+1) + 2 \cdot |x| + |j(x,y)| \cdot |x|$. For $i = 2$ the function j does not depend on y and we can assume that we have $|f(x,y)| \leq |j(x)| \leq c \cdot |x| + c$ for a constant c. Observe that in the third clause of h_1 we have for the bounds on f: $c \cdot |v,w| + c = (c \cdot |v| + c) + (c \cdot |w| + c)$. Hence we can take $q(x,y) = c \cdot (|x|+1) + 2 \cdot |x| + c \cdot |x| + c$. In both cases we can find $b \in \mathcal{F}^i$. ■

The stage $\mathcal{F}^2$ has in the general case no bound for the increase $|j(0,y)| \cdot (|x| + 1)$ coming from the first clause so it does not seem probable that the classes $\mathcal{F}^2$ are closed under limited pair recursion.

Lemma 17 *$Rt, Lt, Left, Right, s, p \in \mathcal{P}^2$.*

Proof: All four functions f are derived by parameterless pair recursion exactly as in the section 3. For bounds we can use $f(x) \leq 0, x$. ■

Lemma 18 $\mathcal{P}^i \subset \mathcal{PM}^i \subset \mathcal{PO}^i$ and $\mathcal{P}^{i+3} = \mathcal{PM}^{i+3} = \mathcal{PO}^{i+3}$.

Proof: To prove the first claim it suffices to show $\mathcal{PM}^i \subset \mathcal{PO}^i$ by proving $\mathcal{PO}^i$ closed under bounded unary minimum. We leave to the reader the definition by unary iteration of a downward search (using p) looking for a number satisfying the predicate. For the second claim it suffices to show $\mathcal{P}^{i+3}$ closed under limited unary iteration. We use the lemmas 13, 16, and the part $\mathcal{PR}_3 \subset \mathcal{PR}_1$ of the theorem 6 under the assumption $|g^x(y)| \leq |j(x,y)|$ where we may assume that j is monotone in the pair size. We observe that $f_1(m,x,y) = x \dot{-} m, g^{\min(x,m)}(y)$ and so we derive a $\mathcal{P}^{i+3}$ bounding condition $|f_1(m,x,y)| \leq |x, j(x,y)|$. ■

It is unknown whether for $i \leq 2.5$ any of the inclusions are strict.

Lemma 19 We have $+, \dot{-}, \cdot, \div, d \in \mathcal{PO}^2$ and $E_i \in \mathcal{PO}^i$ where we denote the unary equivalents of the binary functions by the same symbols (and also use the infix notation where appropriate).

Proof: As already mentioned in the section 3 the unary equivalents of the arithmetic functions are derived by unary iteration following the usual derivation. It suffices to find a $\mathcal{P}^2$ bound on the multiplication as $x + y \leq s(x) \cdot s(y)$. From (3) and the reasoning following it we have

$$2^{|x|} \leq C(|x|+1) \leq \sigma(|x|+2) \leq x \oplus 2. \tag{12}$$

From this and (9) we get

$$x \cdot y \leq 2^{2 \cdot |x| + 2 + 2 \cdot |y| + 2} = 2^{|(x,x) \oplus (y,y) \oplus 2|} \leq (x,x) \oplus (y,y) \oplus 2 \oplus 2.$$

We get $E_2 \in \mathcal{PO}^2$ by explicit definition. An unary equivalent of $E_{2.5}(x,y) = x^{d(y)}$ is obtained by the pair iteration of multiplication in the length $d(y) \leq 2 \cdot |y| + 3 = |1, y, y|$. For a $\mathcal{PO}^{2.5}$ bound it suffices to dominate $x^{|y|}$ where from (11,12) we get $x^{|y|} \leq 2^{|(1,x,x) \otimes y|} \leq (1,x,x) \otimes y \oplus 2$. The functions $E_{i+3}(x) = E_{i+2}^x(2)$ are obtained by unary iterations with $\mathcal{PO}^{i+3}$ bounds obtained from (9) as

$$\begin{aligned} E_{i+2}^x(y) \quad &\leq \quad E_{i+3}(x+y) \leq E_{i+3}(4^{|x+y|+1}) = E_{i+3}|9 \uparrow (0, x+y)| = \\ &|F_{i+3}(9 \uparrow (0, x+y))| \leq F_{i+3}(9 \uparrow (0, x+y)). \ ■ \end{aligned}$$

We will see in the section 5 that the above functions can be derived also in the corresponding classes $\mathcal{P}^i$ but this will not come easily as we will have to detour through binary arithmetic and Turing machine simulation.

Theorem 20 The union of each of the hierarchies $\mathcal{P}^i$, $\mathcal{PM}^i$, and $\mathcal{PO}^i$ is the class of primitive pair recursive functions.

Proof: As the three unions are identical we will show $\bigcup_i \mathcal{P}^i = \mathcal{PR}_2$. $\subset$: Obvious, since $F_i \in \mathcal{PR}_2$. $\supset$: It suffices to show the union closed under pair iteration. Let f be obtained by the pair iteration of $g \in \bigcup_i \mathcal{P}^i$. Hence, for some c and $i \geq 3$: $g \in \mathcal{P}^i$ and $|g(x)| \leq |F_i^c(x)| = E_i^c|x|$. By induction on n we get $|g^n(x)| \leq E_i^{c \cdot n}|x|$ and hence $|g^{|x|}(y)| \leq E_i^{c \cdot |x|} \leq E_{i+1}(c \cdot |x| + |y|) = |F_{i+1}(\sigma(c) \otimes x \oplus y)|$. Thus $f \in \mathcal{P}^{i+1} \subset \bigcup_i \mathcal{P}^i$. ∎

From here until the end of the section we deal with n-ary functions and always indicate pairing by $P(x, y)$. The pairing function P defined by (7) is in $\mathcal{E}^3$ because the functions C and σ are. An inspection of the definition of P reveals that that the elementary functions are applied only to arguments of logarithmic size. The detour can be eliminated:

Lemma 21 The functions P, H, T, and $|x|$ are in $\mathcal{E}^2$.

Proof: We have both d and the *bounded exponentiation* $\min(2^x, y)$ in $\mathcal{E}^0$. Hence $q(x, b) = \min(2^{\min(x, d(b))}, 2 \cdot b + 1) \div 2$ is in $\mathcal{E}^1$. As $2^{d(b)} \leq 2 \cdot b + 1$ we have $d\, q(x, b) = \min(x, d(b))$. We derive the function $C\, d(x) \in \mathcal{E}^2$ by course of values recursion: $C\, d(0) = 1$, $C\, d(x+1) = \frac{4 \cdot d\, k(x)+2}{d\, k(x)+2} \cdot C\, d\, k(x)$ where $k(x) = (x+1) \div 2$. This relies on the recurrent form of C and the fact that $d(x + 1) = d\, k(x) + 1$. An $\mathcal{E}^2$ bound comes from (3) as $C\, d(x) \leq 4^{d(x)} \leq (2 \cdot x + 1)^2$. $\mathcal{E}^2$ is closed under this special form of limited course of values recursion. We note that for $i \leq d(x)$ we have $d\, q(i, x) = i$ and so from (4) we can define the function $\sigma\, d(x) = \sum_{i<d(x)} C\, d\, q(i, x)$ in $\mathcal{E}^2$. We derive $|x|$ in $\mathcal{E}^2$ by modifying (5):

$$|x| = \mu\, n \leq d(2 \cdot x)\, [\, x < \sigma\, d\, q(n + 1, 4 \cdot x)\,]. \tag{13}$$

This is justified by the facts that for $x > 0$: $|x| \leq d(x)+1 = d(2\cdot x)$ and $n \leq d(2 \cdot x) \rightarrow d\, q(n+1, 4\cdot x) = n+1$. The functions $C\, d(x)$ and $\sigma\, d(x)$ serve as a basis for the derivation in $\mathcal{E}^2$ of functions $C(\boldsymbol{a}(x_1, \cdots, x_n))$ and $\sigma(\boldsymbol{a}(x_1, \cdots, x_n))$ whenever we have an $\mathcal{E}^2$ bound $p(x_1, \cdots, x_n)$ such that $\boldsymbol{a}(x_1, \cdots, x_n) \leq d\, p(x_1, \cdots, x_n)$. This is because then $d\, q(\boldsymbol{a}(x_1, \cdots, x_n), p(x_1, \cdots, x_n)) = \boldsymbol{a}(x_1, \cdots, x_n)$. For instance, from $|x| \leq d(2 \cdot x)$ we get $d\, q(|x|, 2 \cdot x) = |x|$ and so the functions $\sigma\, |x|$, $\sigma(|x| + |y| + 1)$, and $C(|x| + |y| - |i|)$ are in $\mathcal{E}^2$. Hence both $Ro(x) = x \dot{-} \sigma\, |x|$ and P defined by the equation (7) are in $\mathcal{E}^2$. The projection function H is in $\mathcal{E}^2$ by bounded minimum: $H(x) = \mu\, v \leq x\, [\, \exists w \leq x\; x = P(v, w)\,]$. Similarly, for T. ∎

Theorem 22 (Characterization of $\mathcal{PO}^i$) The classes $\mathcal{PO}^i$ of the pair order hierachy consist exactly of the unary functions of the classes $\mathcal{E}^i$ of the augmented Grzegorczyk hierarchy.

Proof: $\mathcal{PO}^i \subset \mathcal{E}^i$: By the lemma 21 the classes $\mathcal{E}^i$ are closed under pairing and contain I, H, and T. The function D can be easily derived in $\mathcal{E}^i$ and so the classes are closed under explicit clausal definitions. We can now easily show $\mathcal{E}^i$

closed under both limited iteration and pair iteration with the help of limited recursion. It remains to derive $F_i \in \mathcal{E}^i$. For that we derive $\oplus$ in $\mathcal{E}^2$ just as in the lemma 8. As in the lemma 12 we derive $\otimes \in \mathcal{E}^{2.5}$ with the bound obtained from (9) as $x \otimes y \leq 4^{|x \otimes y|+1} = 4^{|x|\cdot|y|+1} \leq 4^{(d(x)+1)\cdot(d(y)+1)+1}$. The functions F_{i+3} are obtained by pair iteration in $\mathcal{E}^{i+3}$ with the bounds $F_{i+3}(x) \leq \sigma(|F_{i+3}(x)|+1) = \sigma(E_{i+3}|x|+1)$.

For the converse inclusion we show by induction on the construction of $\mathcal{E}^i$ a stronger claim that both the unary functions and the unary equivalents of functions in $\mathcal{E}^i$ are in $\mathcal{PO}^i$. This is certainly true of Z, s, and E_i. The unary equivalents of the projection functions $U_i^n(x_1, \cdots, x_n) = x_i$ are obtained by clausal definitions of the same form. The closure under n-ary composition and limited recursion which is simulated by limited unary iteration is left to the reader. ■

Theorem 23 (Characterization of primitive pair recursive functions) The classes $\mathcal{PR}_i$ consist exactly of unary primitive recursive functions.

Proof: From the theorems 20 and 22 as $\bigcup_i \mathcal{E}^i$ are primitive recursive functions. ■

The operators of recursion and of unary iteration go for too long and permit a direct jump from $\mathcal{E}^2$ to $\mathcal{E}^3$ (from $\mathcal{PO}^2$ to $\mathcal{PO}^3$) by iterating the multiplication. This is why the class $\mathcal{E}^{2.5}$ is skipped in the Grzegorczyk hierarchy. Pair iteration is logarithmically shorter and the jumps by pair iteration go from $\oplus \in \mathcal{F}^2$ to $\otimes \in \mathcal{F}^{2.5}$ and from there to $\uparrow \in \mathcal{F}^3$.

5 Turing Machine Characterizations

We will now characterize some pair-based classes in terms of computational complexity and relate some of the open problems in the complexity theory to those in the small classes. For that we need to know the class $\mathcal{F}^i$ into which a simulation of a particular resource bounded Turing machine falls. So suppose that a given Turing machine M has tape symbols $a_0, a_1, \cdots, a_k$ $(k \geq 1)$ where a_0 is the blank symbol. M has states $q_0, q_1, \cdots, q_m$ $(m \geq 1)$ where q_0 is the initial state and q_1 the terminating state. Let us further assume that M stops for every input word with the length n after at most $t(n)$ steps and uses at most $p(n)$ squares of tape. We assume that $t(n)$ is monotone and $p(n)$ a polynomial. We can assume that M is started by placing the initial word on the otherwise blank tape and make the currently scanned square the first symbol of the word (if any). The machine stops scanning the leftmost non-blank symbol on the tape (if any).

We will code the tape symbol a_i by i and the state q_j by j. Words will be coded by lists consisting of codes of tape symbols. The tape will be coded by two lists l and r where the list r codes the word from the currently scanned square to the right and l the word to the left of the currently scanned square. The list l is reversed so the symbol immediately to the left of the currently scanned square is the first one in l. When $r = 0$ the currently scanned square is a blank at the end of the tape and a move to the right extends the tape by setting $l_1, r_1 = (0, l), 0$. Similarly for l.

It should be obvious that we can define a simple function Mv^M such that $Mv^M(q, l, r) = q_1, l_1, r_1$ holds if one transition of M in the state q and tape configuration l, r results in the new state q_1 and tape configuration l_1, r_1. We can require that r_1 never contains trailing blanks and that $Mv^M(1, l, r) = 1, l, r$ in the terminating state. Let $Mv_1^M \in \mathcal{P}^2$ be the bounded version of Mv^M (see the lemma 15). We set

$$Tm^M(x) = y \leftarrow (Mv_1^M)^{t\,|x|}(b(x), 0, 0, x) = b_1, q_1, l_1, y.$$

We wish to define the function b in such a way that $|(Mv^M)^n(0, 0, x)| \leq |b(x)|$ holds. Then, since $Len(x) \leq |x|$, for a given code x of an input word $Tm^M(x)$ would yield the code of the output word computed by the machine M.

An input word coded by x contains $Len(x)$ symbols each of which has the pair size not exceeding $|k|$. As the length of the tape $Len(l) + Len(r)$ is always bounded by $p\,Len(x)$ we can see that for some q, l and r we have

$$\begin{aligned}&|(Mv^M)^n(0, 0, x)| = |q, l, r| \leq |m| + 1 + |l, r| \leq \\ &\qquad |m| + 1 + (|k| + 1) \cdot p\,Len(x) + 1 \leq |m| + (|k| + 1) \cdot p\,|x| + 2 = |b(x)|\end{aligned}$$

where the function b is obtained by the lemma 9. If $p(n)$ is a linear form then $b \in \mathcal{P}^2$ otherwise $b \in \mathcal{P}^{2.5}$.

Theorem 24 The function Tm^M simulating a Turing machine M computing in (i) linear space, polymomial time is in $\mathcal{P}^2$, (ii) linear space is in $\mathcal{PO}^2$, (iii) polynomial time is in $\mathcal{P}^{2.5}$, and (iv) polynomial space is in $\mathcal{PO}^{2.5}$.

Proof: It should be obvious that the function Tm^M will be in the particular class provided **(a):** the bounding function $b(x)$ is in the same class, and **(b):** we can achieve $t\,|x|$ iterations in there. **(a):** In the cases (i)(ii) the polynomial $p(n)$ is a linear form and so $b \in \mathcal{P}^2$, in the cases (iii)(iv) $b \in \mathcal{P}^{2.5}$.

(b): In the cases (ii)(iv) we can assume that t has the form $t(n) = (k + 1)^{p(n)} \cdot (p(n) + 1) \cdot (m + 1)$. This is because the expression on the right gives a bound on the number of possible configurations of M and the machine must stop before the bound is reached. The number of iterations of Mv_1^M is $t\,|x| = (k + 1)^{p\,|x|} \cdot (p\,|x| + 1) \cdot (m + 1)$. We observe that for a constant c we can define the function $f(x) = c^{|x|}$ in $\mathcal{PO}^2$ by limited pair iteration of multiplication with the bound obtained from (8) as $c^{|x|} \leq (2^{|x|})^{d(c)} \leq (4 \cdot x + 1)^{d(c)}$. We can define $f(x, y) = y^{|x|}$ in $\mathcal{PO}^{2.5}$ by a limited pair iteration of multiplication with the bound $y^{|x|} \leq y^{d(x)+1}$. By an n-fold composition of f we can get $f_1(x, y) = y^{|x|^n}$ in $\mathcal{PO}^{2.5}$. Thus the function $t_1(x) = t\,|x|$ is $\mathcal{PO}^2$ in the case (ii) and $\mathcal{PO}^{2.5}$ in the case (iv). We get Tm^M in the same class by limited unary iteration.

In the cases (i)(iii) the time function $t(n)$ is a polynomial and we wish to achieve $t\,|x|$ iterations of Mv_1^M. This can be done both in $\mathcal{P}^2$ (case (i) when $b \in \mathcal{P}^2$) and in $\mathcal{P}^{2.5}$ (case (iii) when $b \in \mathcal{P}^{2.5}$) by induction on the structure of the polynomial $t(n)$. Let us for simplicity assume that we wish to derive $f(x, y) = g^{t\,|x|}(y)$ by pair iteration. If $t(n) = c$ for a constant c we define $f(x, y) = g^{|\sigma(c)|}(y)$. If $t(n) = n$ we define $f(x, y) = g^{|x|}(y)$. If $t(n) = t_1(n) + t_2(n)$ we set $f(x, y) = g^{t_1|x|}g^{t_2|x|}(y)$. If $t(n) = t_1(n) \cdot t_2(n)$ we set up a 'nested' pair iteration by

$$\begin{aligned}&f(x, y) = y_1 \leftarrow f_1^{t_1|x|}(x, y) = x_1, y_1 \\ &f_1(x, y) = x, g^{t_2|x|}(y).\end{aligned}$$

The reader will see that in our special case there is no problem with bounds. ■

We say that a Turing machine M with three tape symbols: blank, 0 and 1 *computes* an n-ary function $f(x_1,\cdots,x_n)$ when, after giving it the words coding the arguments in the binary and separated by blanks as input, the machine stops with a word coding the number $f(x)$ in the binary as output. In order to convert between numbers and codes of such words we will need two functions B and B^{-1}. The function $B(x)$ yields a list of 1's and 2's called the *binary code* of the number $x>0$, i.e. the list coding the word with the binary representation of x (note that the bits 0, and 1 are coded in our Turing simulation by 1 and 2 respectively). We set $B(0)=0$. Any function B^{-1} such that $B^{-1}B(x)=x$ is a *binary inverse of* B.

Lemma 25 The functions P, H, T, and $|x|$ can be computed by Turing machines in polynomial time and linear space.

Proof: We will be closely following the proof of the lemma 21. For the duration of the proof we indicate the pairing by $P(x,y)$ as we are discussing n-ary functions. For an $\mathcal{E}^2$ function f we have $f(x_1,\cdots,x_n)\le p(x_1,\cdots,x_n)$ where p is a polynomial. Hence $d\,f(x_1,\cdots,x_n)\le p_1(d(x_1),\cdots,d(x_n))$ for a linear form p_1. This means that the $\mathcal{E}^2$ functions defined in the proof of the lemma 21 and below can be all computed in linear space. We have to make sure that this can be done in polynomial time. The operators of recursion, bounded summation, and bounded minimum used in the derivation of $C\,d(x)$, $\sigma\,d(x)$, and $|x|$ respectively have the iteration length $d(x)$ and hence can be computed in time polynomial in $d(x)$.

Bounded summation and minimum in the definitions of P, H, and T are too long and cannot be done in polynomial time. A moment of reflection on the enumeration (1) shows that

$$P(x,y)=P(\sigma\,|x|,\sigma\,|y|)+Ro(x)\cdot C\,|y|+Ro(y). \tag{14}$$

The number $s=P(\sigma\,|x|,\sigma\,|y|)$ is the first number with the head size $n=|x|$ and tail size $m=|y|$. We have $s=\sigma(n+m+1)+\sum_{i<n}C(i)\cdot C(n+m-i)$. Thus $P(\sigma\,|x|,\sigma\,|y|)=\sigma(|x|+|y|+1)+\sum_{i<|x|}C\,|x|\cdot C(|x|+|y|-i)$. The functions derived from $C\,d(x)$ and $\sigma\,d(x)$ by bounds mentioned in the proof of the lemma 21 can be computed in polynomial time and so can the function P. We could compute the projection functions H and T from (14) in polynomial time provided we knew how to compute $|H(x)|$ and $|T(x)|$ in polynomial time as for $x>0$ we could then compute $H(x)=\sigma\,|H(x)|+(x\mathbin{\dot-}P(\sigma\,|H(x)|,\sigma\,|T(x)|))\div C\,|T(x)|$ with a similar identity for $T(x)$. For $x>0$ we can see that $|H(x)|=\max_{n<|x|}[\,P(\sigma(n),\sigma(|x|\mathbin{\dot-}1\mathbin{\dot-}n))\le x\,]$ is computable in polynomial time because for $n\le|x|\le d(2\cdot x)$ we have $\sigma(n)=\sigma\,d\,q(n,2\cdot x)$. The function $|T(x)|$ is computed similarly. ■

Lemma 26 $B,B^{-1}\in\mathcal{P}^2$.

Proof: This would be easy if we had sufficient arithmetic in $\mathcal{P}^2$. We do have s and p but we cannot iterate them sufficiently long in order to obtain $+$ and $x\div 2$. Fortunately, we can proceed by a detour through Turing machines. Let Mp, Mh, and Mt be the Turing machines computing the functions P, H, and T in polynomial time and linear space (they exist by the lemma 25). By the theorem 24 there are $\mathcal{P}^2$ functions $\overline{P}$, $\overline{H}$, and $\overline{T}$ such that $\overline{P}(x,y) = Tm^{Mp}(x \oplus (0,y))$, $\overline{H}(x) = Tm^{Mh}(x)$, and $\overline{T}(x) = Tm^{Mt}(x)$. These functions accept and yield binary codes. We observe that $B(x,y) = \overline{P}(B(x), B(y))$. This, and $B(0) = 0$ constitutes a definition of B by parameterless pair recursion. We have $B \in \mathcal{P}^2$ because

$$|B(x)| \leq 3 \cdot \mathit{Len}\, B(x) = 3 \cdot d(x) \leq 6 \cdot |x| + 9. \tag{15}$$

Let $R(z)$ hold iff z is a binary code, i.e. $z = B(x)$ for some x. We are looking for a function B^{-1} such that $B^{-1}(0) = 0$ and $R(z) \wedge z > 0 \rightarrow B^{-1}(z) = B^{-1}\,\overline{H}(z), B^{-1}\,\overline{T}(z)$. Since $\overline{H}\,B(v,w) = B(v)$ and $\overline{T}\,B(v,w) = B(w)$, we can then show by pair induction on x that $B^{-1}B(x) = x$. Consider the derivation:

$$\begin{array}{l}
B^{-1}(z) = y \leftarrow g^{|t(z)|}(0,(0,z),0) = (y,s), i \\
g(s,(0,0),i) = (0,s), i \\
g(s,(0,z),i) = s,(0,\overline{T}(z)),(0,\overline{H}(z)),0,i \leftarrow z = z_1, z_2 \\
g((x,y,s),0,i) = ((x,y),s), i \\
g(s,0) = s, 0.
\end{array}$$

When B^{-1} is called with a binary code $B(x)$ the value $0, B(x)$ is pushed on the input stack i. The iteration of the function g accumulates the result in the output stack s by breaking down the binary codes of components z of x which are stored on i in the form $(0, B(z))$ The marker 0 is placed on the input stack to mark that the pairing operation should occur on the output stack (the third clause for f). The iteration of g should go once for each 0 and twice for each pair in x, i.e. altogether $3 \cdot |x| + 1$ times. The $\mathcal{P}^2$ function t is obtained by the lemma 9 to satisfy $|t(z)| = 3 \cdot |z| + 4$. We then have

$$3 \cdot |x| + 1 \leq 3 \cdot d(x) + 4 = 3 \cdot \mathit{Len}\, B(x) + 4 \leq 3 \cdot |B(x)| + 4 = |t\, B(x)|.$$

It should be clear that the function B^{-1} satisfies the above conditions and so it is a binary inverse of B. In order to show $B^{-1} \in \mathcal{P}^2$ we intend to use the lemma 15 with $f_1(z) = g^{|t(z)|}i(z)$ for $i(z) = 0,(0,z),0$ to obtain a function $f \in \mathcal{P}^2$ coinciding with f_1 on binary codes. For that we need to show $R(z) \rightarrow |g^n i(z)| \leq |b(z)|$, i.e. $|g^n i\, B(x)| \leq |b\, B(x)|$ for some function $b \in \mathcal{P}^2$. The inspection of the function g reveals that for the initial argument $B(x)$ the only clause which increases the pair size is the second one where the increase is $|B(v)| + |B(w)| + 2 - |B(v,w)|$ for each component v, w of x. This clause is applied $|x|$ times. In estimating the iteration of g we can use the upper bound (15) instead of B. As we have $(6 \cdot |v| + 9) + (6 \cdot |w| + 9) + 2 = (6 \cdot |v,w| + 9) + 5$ we are losing 5 per use of the clause. Hence we are looking for a b such that

$$|g^n i\, B(x)| \leq |i\, B(x)| + (6 \cdot |x| + 9) + 5 \cdot |x| = |B(x)| + 11 \cdot |x| + 12 \leq |b\, B(x)|.$$

Now $|x| \leq d(x) + 1 = \mathit{Len}\, B(x) + 1 \leq |B(x)| + 1$, and so it suffices to obtain the function b such that $|b(z)| = |z| + 11 \cdot (|z| + 1) + 12$ by the lemma 9. ■

Theorem 27 The classes $\mathcal{P}^2$, $\mathcal{PO}^2$, $\mathcal{P}^{2.5}$, and $\mathcal{PO}^{2.5}$ consist exactly of unary functions computed by Turing machines in linear space/polynomial time, linear space, polynomial time, and polynomial space respectively.

Proof: Let the function f be computed by a Turing machine M with one of the above time/space bounds. By the theorem 24 the function Tm^M simulating its operations is in the corresponding pair class. We have $f(x) = B^{-1}\, Tm^M\, B(x)$. The function f is in the same pair class by the lemma 26.

Vice versa, Turing machines operating with the above bounds can certainly compute the functions obtained by composition and pairing from functions computable by machines with similar bounds. The latter holds because of the lemma 25. Such Turing machines can also pair iterate such functions in polynomial time (for $\mathcal{P}^2$, $\mathcal{P}^{2.5}$ functions) because the time is polynomial in $d(x)$ which is on the order of $|x|$. ■

Now we see that the class $\mathcal{P}^2$ contains $+, \dot{-}, \cdot, \div$, etc. The Turing machine characterization of $\mathcal{PO}^2$ can be also proved from the theorem 22 by the result of Ritchie [10] that $\mathcal{E}^2$ consists of n-ary functions Turing computable in linear space. Cobham [2, 11] was the first one to characterize the n-ary functions computable in polynomial time by an inductively defined class based on the *bit size recursion* with the length of iteration $d(x)$.

Let us see why the function $J(x, y) = ((x + y) \cdot (x + y + 1)) \div 2 + x + 1$ obtained from the standard recursion-theoretic pairing function is not a suitable pairing function. It is not difficult to see that the $\mathcal{E}^3$ function $M(n)$ such that $M(0) = 0$ and $M\, s(n) = J(M(n), 0)$ yields the greatest number with the pair size n. A more detailed analysis of J shows that $2^{2^n} \leq M(n + 3)$. We cannot have $d(x) = O(|x|)$ as $2^n \leq d(2^{2^n}) \leq d\, M(n + 3)$ while $|M(n + 3)| = n + 3$. The function J widely disperses the numbers with the same pair size. It can be shown that the function $Lt_J(x) = M\,|x|$ is only elementary. It is hard to see how we can define arithmetic in $\mathcal{P}^2_J$ when we cannot simulate the binary arithmetic by pair iteration as the latter is exponentially shorter than the length of binary codes.

We now characterize the class $\mathcal{PM}^{2.5}_*$. The class of languages PH is the union of the *polynomial hierachy* ([12]). The languages in PH are accepted by alternating Turing machines in polynomial time with constantly many alternations. We have $P \subset NP, coNP \subset PH$. A language $L \subset \Sigma^*$ in PH is such that

$$w \in L \leftrightarrow Q_1\, w_1 \in \Sigma^{p_1\, Len(w)} \cdots Q_n\, w_n \in \Sigma^{p_n\, Len(w)}\, \langle w, w_1, \cdots, w_n \rangle \in L_1 \quad (16)$$

where Q_i is a sequence of alternating quantifiers, $p_i(x)$ polynomials, and $L_1 \in P$. With a unary predicate $R \in \mathcal{F}$ we naturally *associate* a language over $\Sigma = \{0, 1\}$ consisting of words in binary representation of numbers satisfying R. Clearly, a Turing machine computing the predicate R in the above defined sense can be turned into an acceptor accepting the associated language with the same resource bounds, and vice versa.

Theorem 28 The languages over $\{0, 1\}$ in PH are exactly the languages associated with the $\mathcal{PM}^{2.5}$ predicates.

Proof: The classes $\mathcal{PM}^i$, being closed under bounded unary minimum, are closed under bounded unary quantification: $Q(x, y) \leftrightarrow \exists z \leq x\, R(z, y)$ (similarly for $\forall$). In the stage $\mathcal{F}^{2.5}$ we will write the bounds also in the form $\exists |z| \leq p|x|$ for a polynomial $p(n)$. If f is defined in $\mathcal{PM}^{2.5}$ by limited pair iteration of g such that $|f(x)| \leq |j(x)|$ for a monotone in the pair size j then f can be obtained by bounded unary minimum as

$$\begin{aligned} f(x, y) = \mu\, v \leq 0, j(x, y)\, [\, \exists |t| \leq (|j(x, y)| + 1) \cdot |x|\, (\\ Tb(t) \wedge Len(t) = |x| \wedge \exists s \leq t\, (t = (x, s) \oplus (v, 0)))\,] \end{aligned}$$

where $Tb(t)$ is a table containing the iterations of g:

$$Tb(t) \leftrightarrow \forall s_1 \leq t \, \forall s_2 \leq t \, \forall x \leq t \, \forall y \leq t \, (t = s_1 \oplus (x, y, s_2) \rightarrow y = g(x)).$$

By induction on the construction of $\mathcal{PM}^{2.5}$ (we do not have to consider the pair iteration) we can show that to every $f \in \mathcal{PM}^{2.5}$ there is a formula equivalent to $f(x) = y$ consisting of $\mathcal{P}^2$ predicates, connectives, and quantifiers with bounds $|x| \leq p|y|$. This should be seen from the fact that for $f(x) = g(x), h(x)$ we have

$$f(x) = y \leftrightarrow \exists |v| \leq p_1|x| \, \exists |w| \leq p_2|x| \, (y = v, w \wedge g(x) = v \wedge h(x) = w)$$

where the equivalent formulas for g and h are obtained by induction hypotheses and the polynomials are the $\mathcal{F}^{2.5}$ bounds on theses functions. Composition is similar. For f defined by the bounded unary minimum $f(x, y) = \mu z \leq x \, [\, g(z, y) = 1 \,]$ we set

$$f(x, y) = z \leftrightarrow g(z, y) = 1 \wedge z \leq x \wedge \forall v < z \, g(v, y) = 0 \vee z = 0 \wedge \forall v \leq x \, g(v, y) = 0.$$

For a predicate $R \in \mathcal{PM}^{2.5}$ we can then write the equivalent formula for $R_*(x) = 1$ in a prenex form with alternating quantifiers:

$$R(x) \leftrightarrow Q_1 \, |x_1| \leq p_1(|x|) \cdots Q_n \, |x_n| \leq p_n(|x|) \, S(x, x_1, \cdots, x_n)$$

where $S \in \mathcal{P}^2$. The rest of the proof then follows from the correspondence to (16) and from the theorem 27. ∎

We probably cannot derive limited pair iteration by bounded unary minimum in the class $\mathcal{PM}^2$. This class but without pair iteration is the class $\mathcal{M}^2$ of the minimum hierarchy of Harrow whose predicates are *constructive arithmetic*, or *bounded arithmetic* predicates (see [8, 11]).

Theorem 29 $P = NP$ iff $\mathcal{P}^{2.5}$ is closed under bounded unary minimum, i.e. $\mathcal{P}^{2.5} = \mathcal{PM}^{2.5}$. $P = PSPACE$ iff $\mathcal{P}^{2.5}$ is closed under limited unary iteration, i.e. $\mathcal{P}^{2.5} = \mathcal{PO}^{2.5}$. $PH = PSPACE$ iff $\mathcal{PM}^{2.5}$ is closed under limited unary iteration, i.e. $\mathcal{PM}^{2.5} = \mathcal{PO}^{2.5}$.

Proof: If $P = NP$ then the polynomial hierachy collapses: $P = PH$ (see [9]) and then $\mathcal{P}^{2.5}_* = \mathcal{PM}^{2.5}_*$. Vice versa, if the latter identity holds then by the theorems 24, 27, and 28 we have $P = PH$ and hence $P = NP$. Similarly, $P = PSPACE$ iff $\mathcal{P}^{2.5}_* = \mathcal{PO}^{2.5}_*$ and $PH = PSPACE$ iff $\mathcal{PM}^{2.5}_* = \mathcal{PO}^{2.5}_*$.

Clearly, from $\mathcal{F}_1 = \mathcal{F}_2$ follows $(\mathcal{F}_1)_* = (\mathcal{F}_2)_*$. So it remains to show the converse where we assume $(\mathcal{F}_1)_* = (\mathcal{F}_2)_*$ for the above three pairs of relation classes. We have $\mathcal{F}_1 \subset \mathcal{F}_2$. Let us denote by $(a)_i$ the $\mathcal{P}^2$ function yielding the i-th element $(0 \leq i)$ of the list a or 0 if $i \geq Len(a)$. Take any function $f \in \mathcal{F}_2$. The function f is bounded by a function $j \in \mathcal{F}_1$. The relation $R(x, i) \leftrightarrow (B \, f(x))_i = 1$ is in $(\mathcal{F}_2)_*$ and hence also in $(\mathcal{F}_1)_*$. Now, $Len \, B \, f(x) \leq Len \, B \, j(x)$ and we can derive by pair iteration a function $k \in \mathcal{F}_1$ such that $k(x) = B \, f(x)$ by assembling the bits of the binary code with the help of the predicate R. Hence $f(x) = B^{-1} \, k(x)$ is in $\mathcal{F}_1$ and we have $\mathcal{F}_1 = \mathcal{F}_2$. ∎

6 Conclusions

Some of the problems in the small recursive classes have been open for over forty years now. We think that they did not receive sufficient attention because they were not the central problems of hierarchy theory. With our characterizations the problems are shown central to the computer science. We believe that a solution of any of the open problems in the small classes will be by a technique directly applicable to the solution of the $P = NP$ problem.

Although the central results of this paper are contained in the characterization theorems (22,27,28,29) we would like to remind the reader that this research was started as a search for a feasible declarative programming language based on subrecursion. We plan to develop and implement on computers our clausal language in the near future.

References

[1] P. Borovansky, P. J. Voda. Types as Values Polymorphism. In proceedings of SOFSEM conference 1993.

[2] A. Cobham. The intrinsic computational difficulty of functions. In Proc. Int. Conf. Logic, Meth. Phil.(ed Y. Bar Hillel), 24-30, North Holland, Amsterdam, 1965.

[3] M. Davis. Computability and Unsolvability, McGraw Hill, New York. 1958.

[4] S. Feferman. Finitary Inductively Presented Logics. In Logic Colloquium, North Holland 1989.

[5] N. D. Jones. Constant Time Factors Do Matter. In proceedings of 25th ACM STOC 93. ACM 1993.

[6] R. L. Graham, D. F. Knuth, O. Patashnik. Concrete mathematics. Addison-Wesley 1989.

[7] A. Grzegorczyk. Some classes of recursive functions. Rozprawy Mate. No. IV, Warsaw 1953.

[8] K. Harrow. Small Grzegorczyk classes and limited minimum. Zeit. Math. Logik, 21, 417-26, 1975.

[9] K.R. Reischuk. Einführung in die Komplexitätstheorie. B.G. Teubner Stuttgart, 1990.

[10] R.W. Ritchie, Classes of predictably computable functions. Trans. Am. Math. Soc. (106), 139-73, 1963.

[11] H.E. Rose. Subrecursion, Functions and Hierarchies. Clarendon Press, Oxford 1984

[12] L. Stockmeyer, The Polynomial-Time Hierachy, Theor. Comp. Sci. 3, 1-22, 1977.

[13] P. J. Voda. Types of Trilogy, Proceedings of the Fifth International Conference on Logic Programming, MIT Press, Cambridge MA, 1988.