Subrecursion as a Basis for a Feasible Programming Language

Paul J. Voda

Institute of Informatics, Comenius University Bratislava, Mlynská dolina, 842 15 Bratislava, Slovakia.
voda@fmph.uniba.sk

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 a pairing function because in this way we can explore the amazing coding powers of S-expressions of LISP within the domain of natural numbers. We introduce three Grzegorczyk-like hierarchies based on pairing and characterize them both in terms of Grzegorczyk hierarchy and computational complexity.

1 Introduction

The motivation for this research comes from our search for a good programming language where we are constructing 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) [9, 1]. 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. The connection between recursive classes of functions (based both on the successor recursion and on the recursion on notation) and classes of computational complexity is quite well-understood now (see for instance [10]). We investigate this connection by recursion based on pairing.

Section Sect. 2 introduces the pairing function $P$. We order the presentation of primitive pair recursive functions in Sect. 3 in such a way that we can quickly
develop computational complexity classes in Sects. 4 and 5. The development critically depends on the right choice of a pairing function. The proofs of most lemmas in this paper are omitted. Interested reader can obtain a copy of the full paper from the author by email.

Our contributions are (i) in the design of a clausal language for the definition of recursive functions essentially as a usable computer programming language (Sect. 3), (ii) in the insight that the natural measure of S-expressions, i.e. the number of pairing operators (\texttt{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 hierarchies (Sect. 4) and Turing machines (Sect. 5). Finally, we get new (iii) 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 \( \mathbb{N} \). It is well known that in the presence of a pairing function we can restrict our attention to the 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 \((\cdot,\cdot)\) is a \textit{semi-suitable pairing function} if it is (P1): a bijection from \( \mathbb{N}^2 \) onto \( \mathbb{N} \setminus \{0\} \), and we have (P2): \((x, y) > x\) and \((x, y) > y\).

The condition (P1) assures the \textit{pairing property} that from \((a, b) = (c, d)\) we get \(a = c\) and \(b = d\) and the property that 0 is the only \texttt{atom}, i.e. the only number not of the form \((x, y)\).

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

**Proof.** By complete induction on \( R(x) \).

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

**Proof.** By pair induction where \( R(x) \) iff \( x \) has a unique representation.

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

Thms. 1 and 2 guarantee that for a semi-suitable pairing function (a): every number \( x \) is either 0 or it can be uniquely 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 \texttt{lists} in the computer science), and vice versa. (b): There exist unique \textit{pair size} \(|x|\) and \textit{length} \( \text{Len}(x) \) functions such that \(|0| = 0, |x, y| = |x| + |y| + 1, \) and \( \text{Len}(0) = 0 \) and \( \text{Len}(x, y) = \text{Len}(y) + 1 \).