On the Relation between Grossone Methodology and Diagonalization of Programs

Emanuele Covino
Dipartimento di Informatica
Università degli Studi di Bari
Bari, Italy
e-mail: emanuele.covino@uniba.it

Antonella Falini
Dipartimento di Informatica
Università degli Studi di Bari
Bari, Italy
e-mail: antonella.falini@uniba.it

***Abstract*—We argue that there could be a connection between the Grossone methodology, introduced by Sergeyev, and the Diagonalization operator defined in one of our previous works. The two concepts are based on similar theoretical principles, and they both deal with the problem of defining infinite objects (numbers or hierarchies of functions, respectively) with a finite number of operators.**

Keywords-grossone methodology; constructive diagonalization.

I. Introduction

Starting from year 2000, one of the Authors has investigated the link between fragments of programming languages and complexity classes [1]–[3]; these works follow the principles of the Implicit Computational Complexity theory from the 1990s. In traditional complexity, classes are defined by imposing explicit resource bounds on Turing machine's computations, but no restrictions are set on the algorithms they execute. On the other hand, implicit complexity studies classes of languages that are defined without any explicit limitation on resources, but rather by imposing suitable linguistic constraints on the way the related algorithms can be written: this allows to obtain programming languages that capture well-known complexity classes, providing what is called an implicit characterization. Within this framework, we were able to define a hierarchy of classes of programs with complexity between polynomial-time $O(n^k)$ and exponential-time $O(n^{n^k})$, using *safe composition* and *safe recursion*, and introducing a new *diagonalization* operator (they will be outlined in Section II).

Within the same timeframe, Sergeyev's work [4]–[6], focused on a new methodology of computation that allows to handle finite, infinities, and infinitesimals quantities; it is of particular interest the discussion about the relation between mechanical computation (the object of the study) and the related description (the language we use to describe it), and how the latter influences the accuracy of the obtained results. We believe that there is an intrinsic analogy between Sergeyev's *grossone* (the numeral that indicates the number of elements of the set $\mathbb{N}$ of natural numbers) and the *diagonalization* (the operator we use to jump outside an infinite sequence of classes of programs). They both share some *constructive* features: grossone is handled as a number (more precisely, as a new numeral used to describe both infinities and infinitesimals), and diagonalization is, as a matter of fact, a well defined program. They both cope with the finite description of infinite objects (numbers or classes of programs).

In Section II, we recall the definitions of safe recursion, diagonalization, and of the hierarchy of programs that can be defined using these two operators. In Section III, we recall the postulates on which the grossone methodology is based. In Section IV, we provide an insight on the previous analogy, and we outline what we consider a research path on the matter.

II. Safe recursion, diagonalization and a hierarchy of programs

In previous papers, we analyzed the computational complexity of programming languages obtained by imposing some linguistic/syntactical restrictions to their definition [2][3]. We have been able to capture a slow-growing hierarchy of programs with computing time between polynomial $O(n^k)$ and exponential $O(n^{n^k})$; this was achieved by means of two well known operators, namely *safe composition* and *safe recursion*, and introducing a new *diagonalization* operator.

Definitions of significant complexity classes have been obtained by Simmons [7], Leivant [8][9], Bellantoni and Cook [10], and Arai and Eguchi [11], among many others. In these papers, the class of polynomial-time computable functions is characterized by means of different versions of *predicative recursion* [10] or *ramified recurrence* [9], starting from a set of initial functions. A predicative definition of a recursive function is based on the idea that no explicitly bounded schemes are used, and that functions have two kind of variables: those whose values are known entirely (and which can be recursed upon), and those whose values are still being computed. These two types of variables are called *safe* and *normal* in [10] (*dormant* and *normal* in [7]); roughly speaking, normal variables are used only for recursive calls, while safe variables are used only for substitution. This separation allows to discard explicitly bounded schemes (e.g., the limited recursion) to characterize classes of functions with a given complexity (e.g., Polytime).

Our version of the *safe recursion* scheme on a binary word algebra is such that $f(x, y, za) = h(f(x, y, z), y, za)$, in which x, y and z are the auxiliary variable, the parameter, and the principal variable, respectively. In our language, the program h cannot re-assign the recursive call $f(x, y, z)$ to the principal variable z. This implies that we always know the number of recursive calls of the step program in a recursive definition: z turns out to be a *dormant* (or *safe*) variable. Starting from a natural definition of constructors and destructors over lists of binary words, we define the hierarchy of classes of programs $\mathcal{T}_k$, with $k \geq 1$, where programs in $\mathcal{T}_1$ can be computed

within linear time, and programs in $\mathcal{T}_{k+1}$ are obtained by one application of safe recursion to programs in $\mathcal{T}_k$; we prove that programs defined in $\mathcal{T}_k$ are exactly those programs computable within time $O(n^k)$.

In order to jump outside this infinite hierarchy of poly-time programs, an operator of *constructive diagonalization* is introduced, extending the previous hierarchy to $\mathcal{T}_\lambda$, with $\omega \leq \lambda \leq \omega^\omega$ (we follow [12] for definitions of structured ordinals). Programs in $\mathcal{T}_{\alpha+1}$ are obtained by one application of safe recursion to elements in $\mathcal{T}_\alpha$. If λ is a limit ordinal, and $\lambda_1, \ldots, \lambda_k, \ldots$ is the associated fundamental sequence, programs f in $\mathcal{T}_\lambda$ are obtained by *diagonalization* on the previously defined sequence of classes $\mathcal{T}_{\lambda_1}, \ldots, \mathcal{T}_{\lambda_k}, \ldots$, if $f(s,t) = \text{ITER}^{|t|}(f_{\lambda_{|t|}})(s,t)$, with

$$\begin{cases} \text{ITER}^1(p)(s,t) & = & \text{ITER}(p)(s,t) \\ \text{ITER}^{k+1}(p)(s,t) & = & \text{ITER}(\text{ITER}^k(p))(s,t), \end{cases}$$

and $f_{\lambda_{|t|}}$ belongs to a previously defined class $\mathcal{T}_{\lambda_{|t|}}$. $\text{ITER}(p)(s,t)$ is a simplified version of our safe recursion, that iterates the function p on s for $|t|$ times.

A program defined by diagonalization still abides by the predicative principle, given that such a program is totally constructive, it has no explicit bounds, and it basically enumerates and iterates programs that already belong to lower-level classes of programs, which are totally defined. This allows us to harmonize in a single hierarchy the classes of programs with computing time bounded by polynomial time $O(n^k)$ and exponential time $O(n^{n^k})$, for each finite k. For instance, at level ω, we select (and iterate i times) programs in the classes $\mathcal{T}_i$, where i is the length of the input; thus, the first level of diagonalization captures the class of all programs whose computation is bounded by a polynomial. By extending this approach to the next levels of structured ordinals, we were able to reach the machines computing their output within exponential time n^{n^k}.

III. GROSSONE CONCEPT

In the first years of 2000's, Yaroslav D. Sergeyev initiated a new methodology of computation in order to provide new ways of computing with infinities and infinitesimals [4]–[6]. Their approach is based on three postulates:

Postulate 1. *There exists infinite and infinitesimal objects but human beings and machines are able to execute only a finite number of operations.*

Postulate 2. *We shall not tell what are the mathematical objects we deal with; we just shall construct more powerful tools that will allow us to improve our capabilities to observe and to describe properties of mathematical objects.*

Postulate 3. *The principle 'The part is less than the whole' is applied to all numbers, (finite, infinite, or infinitesimal), and to all sets and processes (finite or infinite).*

These postulates set the basis for a new way of looking at and measuring mathematical objects, and have been applied in [4], where a new numeral system has been introduced; this system gives the possibility of dealing with numerical computations with finite, infinite, and infinitesimal numbers. In order to do this, a new infinite unit of measure has been introduced as the number of elements of the set $\mathbb{N}$ of natural numbers; it is denoted by the numeral ① , and it is called *grossone*. The Infinite Unit Axiom is what formally defines ①, and it states the principles of infinity, identity, and divisibility, as follows:

1) *infinity*: for any $n \in \mathbb{N}$, it follows that $n < ①$;
2) *identity*:
   - $0 \cdot ① = ① \cdot 0 = 0$;
   - $① - ① = 0$;
   - $\frac{①}{①} = 1$
   - $①^0 = 1^① = 1$
3) *divisibility*: for any $n \in \mathbb{N}$, the numbers $\frac{①}{n}$ are the number of elements of the n^{th} part of $\mathbb{N}$.

It is important to underline that grossone is different from Cantor's ω; due to its finite nature, grossone could be compared to our constructive diagonal operator. In [5], the reader can find a number of applications of Sergeyev's method to different areas (see [13][14] for the application of grossone methodology to the theory of computation).

IV. GROSSONE'S POSTULATES AND DIAGONALIZATION: FUTURE WORK

It is interesting to notice that the Postulates recalled in Section III do not have to be conceived as axioms in a new axiomatic system, but rather they set a methodological basis. They allow to observe and describe mathematical objects and quantities at a different level of refinement, when compared to the standard ways to handle them.

The operators of safe recursion and diagonalization introduced in Section II, together with the related hierarchy, are based on the same principles. First, there exists a potentially infinite number of programs into each class, but we use only two defining tools to capture them. Then, different ways of combining the operators allow us to deal with different hierarchies of program's classes. In our previous work, we analyze a *slow-growing* hierarchy, but we are able to characterize different complexity classes by tweaking the definition and/or the combination of the operators (for instance, by changing the way the diagonalization iterates the functions). In this sense, we are constructing tools that allow us to observe more (complexity) properties of the hierarchy.

Another analogy resides in the nature of grossone and diagonalization, because they both share the same constructive features. Grossone is defined and treated as a number (it is a new numeral used to describe both infinities and infinitesimals), and can be used into effective computations. Diagonalization is actually a well defined program that enumerates and iterates programs at a lower level of complexity; it is used to define new classes of programs that could not be defined using the safe recursion only. They both cope with the problem of describing infinite objects with finite operations.

For the previous reasons, we believe that the relation between grossone methodology and diagonalization of programs is worth exploring.

REFERENCES

[1] S. Caporaso, G. Pani, and E. Covino, "A predicative approach to the classification problem", *Journal of Functional Programming*, vol. 11, no. 1, pp. 95–116, 2001.

[2] E. Covino and G. Pani, "Diagonalization and the complexity of program", in *The Ninth International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking, (COMPUTATION TOOLS 2018), February 18 - 22, 2018 – Barcelona, Spain*, 2018, pp. 1–5.

[3] E. Covino and G. Pani, "Diagonalization and Elementary Complexity", in *The 14th International Conference on Foundations of Computer Science (FCS'18), July 30 - August 2, 2018, Las Vegas, USA*, 2018, pp. 3–9.

[4] Y. D. Sergeyev, "Arithmetic of Infinity", in. Edizioni Orizzonti Meridionali, 2003.

[5] Y. D. Sergeyev, "Numerical infinity and the infinity computer", 2004, Accessed: Mar. 4, 2026. [Online]. Available: http://www.theinfinitycomputer.com

[6] Y. D. Sergeyev, "A new applied approach for executing computations with infinite and infinitesimal quantities", *Informatica*, vol. 19, no. 4, pp. 567–596, 2008.

[7] H. Simmons, "The realm of primitive recursion", *Arch.Math. Logic*, vol. 27, no. 2, pp. 177–188, 1988.

[8] D. Leivant, "Stratified functional programs and computational complexity", in *Proceedings of the 20th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, (POPL'93), Charleston*, 1993, pp. 325–333.

[9] D. Leivant, "Predicative recurrence and computational complexity I: word recurrence and polytime, in Feasible Mathematics II, P.Clote and J.Remmel (eds)", in. Birkauser, 1994, pp. 320–343.

[10] S. Bellantoni and S. Cook, "A New Recursion-Theoretic Characterization Of The Polytime Functions", *Computational Complexity*, vol. 2, pp. 97–110, 1992.

[11] T. Arai and N. Eguchi, "A new function algebra of EXPTIME functions by safe nested recursion", *ACM Transactions on Computational Logic*, vol. 10, no. 4, pp. 1–19, 2009.

[12] M. Fairtlough and S. Weiner, "Hierarchies of provably recursive functions, in Handbook of Proof theory, B. Samuel (ed), Studies in logic and the foundations of mathematics, vol. 137", in. Elsevier, Amsterdam, 1998, Chapter 3, pp. 149–207.

[13] Y. D. Sergeyev and A. Garro, "Observability of Turing Machines: a refinement of the theory of computation", *Informatica*, vol. 21, no. 3, pp. 425–454, 2010.

[14] Y. D. Sergeyev and A. Garro, "The Grossone methodology perspective on Turing machines", in *Automata, Universality, Computation*. A. Adamatzky (ed.), Springer Series "Emergence, Complexity and Computation", 2015, vol. 12, pp. 139-169.