Understanding computing through defining its lexicon[*]

Boyko B. Bantchev

Important questions regarding what computing is and how it pervades nature, science, engineering and other fields remain unanswered. Even restricted to programming or using computer-based technology, computing lacks proper understanding. We argue that one way to bring this understanding closer is by discovering the most fundamental structures and relations in computing and thus building a dictionary of computing.

Doing $\neq$ understanding

Computing, as a human activity, has its origins in times immemorial. Initially it was almost exclusively concerned with doing numeric calculations. With time, logic, geometry, and other formal disciplines have contributed to extending the domain of application of concepts like 'procedure' and 'method' to other areas. It took many centuries before the very notions of computing and computability, information and complexity, data and algorithm became topics of academic scrutiny of their own.

The advent of automatic computers lead to the employment of computing methods and computer programs in a vast number of human activities. This required, and did lead to the discovery of important computing paradigms now embodied in our programming models and languages, software construction methods, and patterns of interfacing of that software to humans. Computing is establishing itself as an inseparable ingredient of our culture – and one that pervades more and more of it.

This new and growing presence challenges our comprehension of computing as a phenomenon, its facets, forms, implications, and related concepts. In order to manage the ever growing diversity and complexity of software, its construction and use, and moreover, in order to enjoy the benefits of the mutual fertilization between computing and other fields, it is necessary that computing is not perceived as the phenomenology of using a computer, or restricted to the sphere of human activities. Computing is any process that has to do with information, and is not only done by humans but is something that occurs in any living system, and any material system at all.

However, this more general understanding has only recently begun to receive wide appreciation. For too long computing was a technology-related concept. Even the science of computing is usually called 'computer science', thus further aggravating the misconception. For the said and other reasons, we are lacking understanding of the profound principles on which computing is based, of what is computable, and of what computing itself and its inherent structure are. Put shortly – if somewhat bluntly – we, as computing scientists, have not yet properly defined our field of study, and we are urged to do so.

In order to observe how inadequate our current perception of computing is we need to look no further than programming, which we have partitioned into 'styles' or 'paradigms'. Two styles, say imperative and functional, can be so much different that it is hard to

[*]The paper has been published in *Mathematics and Education in Mathematics, 2010.* Proc. 39th Spring Conf. of the Union of Bulgarian Mathematicians, April 2010, pp. 171-177

explain how they both express the same general idea of computation. The different styles appear to be almost totally disunited because each one is based on its own set of concepts that barely intersects with that of the other styles.

In fact, it is very difficult even to formulate what a programming style in the above sense is. It is also hard to speculate on what paradigms beyond the known ones are possible: how could we do that in the absence of a language suitable for expressing such ideas? Clearly, we need an underlying language – the true language of computing – that would provide us with the common basic terms in which all aspects of computing could be discussed.

The deeper understanding of computing can:

- improve our abilities to develop and use computer-based technology, in particular to find better ways to approach programming and software construction;
- make us more able to apply computational concepts, such as algorithms and data organisation, to other areas of human activity, not necessarily involving the use of computers;
- help us identify and explore the ways in which computing takes place in Nature.

It is our belief that the road to this deeper understanding passes through discovering the most fundamental structures and relations in computing – those that are neither based on particular models of computing, nor dependent on the current computer technology, but are truly essential and general – and building a dictionary that captures the meanings of the relevant notions.

General views of computing

Computing is seeking its true place in our knowledge system. Recognition of the need for more comprehensive understanding of computing has taken the form of new conceptual paradigms such as 'computational thinking' and 'great principles of computing'.

Historically, there were several attempts to untie the understanding of computing from the restrictive view that bonds it to technology and software. The most recent one is the so called 'computational thinking' (CT) movement, first proclaimed in [12], and also discussed in [3, 13, 14]. CT is aimed at awakening awareness of the role algorithmic structures have or could have in a variety of human activities, from everyday life to science disciplines and manufacturing. According to [12], CT is 'the study of computation – what can be computed and how to compute it'. The cited article goes on to describe CT's characteristic features as follows:

- conceptualization (thinking at multiple levels of abstraction);
- fundamental, not rote skill;
- human-like (imaginative) thinking;
- complementing and combining mathematical and engineering thinking;
- emphasizing computational concepts rather than artifacts;
- for everyone and permeating every human endeavour.

Biology, physics, chemistry, economics are considered successful examples of computational thinking, more precisely – of infiltration of computational methods into science. There can be also an influence in the reverse direction, e. g. new computational paradigms can be extracted from observing the mechanisms for manipulating information in living organisms [10].

In the context of CT, [9] draws attention to the pre-existence of computing concepts in people before they are exposed to education in computing, and to whether programming languages should be redesigned in order to make computing and programming accessible to a wider audience.

Of course, ideas similar to computational thinking are not new. A. Perlis, E. Dijkstra and others maintained that understanding a wide variety of topics could and should be recasted in terms of computation. A. Ershov's vision of the inevitable ubiquity of what has come to be called after him 'second literacy' [7] was in the same vein.

More recently B. Chazelle, in his brilliant essay [4], argued that algorithms are the heart of computing and 'the new language of science', a way of thinking with a revolutionary impact on it. In his words, 'not only is this new "order" empowering the e-technology [. . .] it is also challenging what we mean by knowing, believing, trusting, persuading, and learning'.

F. Brooks in his classic work [2], contrary to Chazelle, although in a narrower context, insists that 'representation is the essence of programming', meaning that (in programming) data structures are of primordial importance and algorithms are secondary.

To us, the stark contradiction between emphasizing algorithms or data as the essence of computing – in whatever understanding of the word – only indicates that they are equally fundamental. Both algorithm and data possess structure and, more importantly, their intimate interrelation is what turns mere structure into computation.

Somewhat earlier than the emergence of the CT movement, P. Denning pioneered the idea of 'great principles of computing' [5, 8], aiming at exposing, naming and describing recurrent computational patterns in both human practice and natural phenomena. Specifically – and in contrast to CT – Denning places explicit emphasis on the fact that computing is a natural phenomenon itself, and therefore occurs virtually everywhere regardless of whether its manifestations are being observed or not ('computation is more fundamental than computational thinking') [6].

As currently defined in [8], the basic principles of computing are categorized in seven major groups: computation, communication, coordination, recollection, automation, evaluation, and design. However, this discrimination is just for convenience of reference; grouping, as well as the actual principles, are considered variable, pursuant to the perfection of those who identify and describe them.

Denning's approach to understanding computing is characterised by accenting on the need for 'developing a new language for discussing the core principles of computing'. This is very much in consonance with our own vision of the subject, although Denning's treatment appears to be significantly different from ours. Our approach and his can be considered complementary.

The intersection of computing and science attracts increasingly more attention. An international expert group produced an extensive report on the topic [11], in which 'computer science' is foreseen to 'make a major, if not reforming contribution to natural sciences' and 'become fundamental' to them. By employing computing techniques, sciences are able to conduct new kinds of experiments that are also generating new kinds of data – of giant complexity and volume. Being able to represent, interpret, and make use of such data is a challenge reflecting back to computing science, and it is anticipated that this challenge may result in revolutionary changes to computing itself.

Computing and programming

Computing is processing information. Although it means more than programming or using computers, in understanding and describing it we have no better place to start than the science of programming, especially programming languages. Indeed, a programming language is an embodiment of a set of ideas – mathematical concepts as well as pragmatic considerations – about designing software, and therefore about the essence of computation.

We create computing systems by programming, and programming and other formal languages are the formalization of our computational thoughts. Each such language offers a set of concepts and mechanisms intended for expressing computations. Not only are these languages a means of representing ideas about computation, but they are the only means so far.

In recognition of this role of programming languages, [14] for example maintains that successful development of computational thinking depends on studying programming and programming languages and using the latter as an environment for growing new computational concepts. In this respect, the work of K. Iverson on the properties of programming notation is found particularly inspiring for its abundance of ideas spanning not only programming but also mathematics.

The work done on designing and using programming languages has already had, and will continue to have, a great impact on clarifying the concepts of computing. Elaborate conceptual systems have been developed within languages, and sometimes across them, to represent views on the subject.

The usefulness of programming languages for conveying computational ideas is unquestionable, but it is also insufficient. In programming, being specific is inescapable, or at least this is how today's programming works. More than in mathematics, utmost precision of meaning is needed, without which the process that a program realises would not have been properly defined, and therefore interpreted. A language represents a specific point of view on computing, but being specific also means being limited in the scope of what can be expressed. The notions defined in a programming language are relatively small in number, restricted to mean what the language needs them to mean, and do not necessarily correspond to similar, or similarly named, notions in other languages. Even this alone creates confusion in understanding computing.

Using terminology from different programming paradigms is another source of confusion, as is the very existence of these similarly not comparable paradigms. Currently, we are not able to answer questions such as: Why do we have exactly these paradigms? Are they natural, in some sense, or do some of them just result from speculation instead of being rooted at the essence of computing? Can we expect new major paradigms to emerge? If so, can we predict them? Is it possible to reconcile all computing paradigms with each other? The inability to tackle such questions clearly indicates the immaturity of our conceptualizing of computing.

Moreover, it is necessary to distinguish essential concepts within a programming language from accidental ones with respect to its programming model, or with respect to computing in general. Languages have too many accidental concepts that only confuse the true understanding of computing.

The language of computing

Previously [1], we have discussed the importance of general and abstract comprehending of structure in programming and gave general formulations of some kinds of structural relationship. The present endeavour can be seen as a continuation and further generalization of that work. For lack of space, we do not discuss in detail particular concepts in computing, and only concentrate on setting the principles on which we see necessary to found building a general dictionary of computing. We intend to present a more elaborate discussion on a number of computing-related concepts in a subsequent publication.

How can we discover the fundamental language of computing? Our proposal rests on the following key principles:

Action- and structure-relatedness.

Any computation consists of actions and has structure. The objects and systems that take part in the computation also have structure. Action and structure are thus the fundamental notions of computing, around which the language of computing should be built. In other words, the dictionary of computing is an enumeration – or a taxonomy – of action- and structure-related phenomena.

As structure is also characteristic for other phenomena and disciplines, what particular kind of structure is that of computing? The 'specificity' of computing's structure is that it is the archetypal structure – that of systems in the general sense, or *the* structure. If computing, i. e. information processes pervade all phenomena, then computing's structure pervades any other form of structure. For example, physical processes may be characterised by the interaction of physical forces and how a system is changed under it. Similarly for chemical or biological processes. But to the extent these systems manifest information-related, i. e. computational properties, they are also characterisable as abstract systems, and thus exhibit structure pertaining to computing. That is the kind of structure that should interest us.

Defining major topics.

Building a general terminology of computing should be done by clustering the needed concepts around properly selected major topics. These would be the major areas and ways in which computing structure manifests itself. Here is an example list of topics.

– Action and closely related concepts, such as subject, resource, access.
– Creation and destruction. Instantiation, reproduction and others also belong here.
– Aggregation and interrelation. Many concepts of varying generality fall into this group, e. g. precedence, nesting, determinacy, tupling, sequencing, bifurcation, looping, subordination, coordination.
– Transformation. This concerns structure in general and can include concepts reflecting various specific kinds of transformation, such as adapting, preserving, complementary etc.
– Interaction. Simple examples of concepts that belong here are invocation, resumption, and interruption.

The following two are perhaps more on the side of the language than computing itself, but are nevertheless relevant.

– Binding. This describes how a resource gets scoped to a particular context of use, and how different scoped entities interact.
– View creation. This includes concepts related to how a computing system is being interpreted, examples being projection and various forms of abstraction.

Generalizing fundamental concepts from programming languages.

Concepts that admit of generalizing should be identified and their most general meaning must be found. Whenever two or more such concepts lead to similar definitions, they probably should be further generalized. For example, in many programming languages an action can be applied uniformly across the elements of a data structure. Also, a binding of some name may be made known within the scope of some set of actions in a program. Both mentioned are instances of *distribution* of a resource across recipients. In the former case, the resource is an action, and in the latter it is a binding, but the structural relation is nevertheless the same.

Looking for computing-related words in the human language(s).

Programming language terminology has already employed a number of words from the English or other natural languages for the purpose of describing computational phenomena. We feel that other such words just need to be discovered. In this way we could draw on the contextual power of words that they bring with themselves in order to enrich the meaning of known computational concepts or even to discover new concepts.

This has been done in the process of creation of programming languages in the past, of which ALGOL 68 is a notable example. There is no reason why it cannot continue in the more general setting of defining computation.

Uniform treatment of actions and data.

Actions and data both carry (computational) structure. However, other than the ability of some programming languages to treat data as actions or vice versa, the structural interrelation between the two is very little explored. It seems that such exploration could be beneficial to better understanding of computing. A number of mathematical disciplines provide us with inspiration for this by successfully making use of dualities: between points and lines or planes in geometry, between nodes and edges in graph theory etc.

References

[1] Bantchev B. B. *Towards a framework for comprehending structure in programs.* Mathematics and Education in mathematics, 1998, pp. 210-215.

[2] Brooks F. P. *The mythical man-month: essays on software engineering*, 2nd ed. Addison-Wesley Professional, 1995.

[3] *Center for computational thinking at Carnegie Mellon*, `http://www.cs.cmu.edu/~CompThink`

[4] Chazelle B. *The Algorithm: idiom of modern science.* `http://www.cs.princeton.edu/~chazelle/pubs/algorithm.html`, 2006.

[5] Denning P. J. *Great principles in computing.* Comm. ACM. Vol. 46 (2003), No.11, pp.15-20.

[6] Denning P. J. *Beyond computational thinking.* Comm. ACM. Vol. 52 (2009), No.6, pp.28-30.

[7] Ershov. A. P. *Programming, the second literacy*, a talk at the 3rd IFIP and UNESCO World Conf. on Computers in Education, 1981. Web: `http://ershov.iis.nsk.su/russian/second_literacy/article.html`.

[8] *Great principles in computing Web site.* `http://greatprinciples.org`.

[9] Guzdial M. *Paving the way for computational thinking.* Comm. ACM. Vol. 51 (2008), No. 8, pp. 25-27.

[10] Priami C. *Algorithmic systems biology.* Comm. ACM. Vol. 52 (2009), No. 5, pp. 80-88.

[11] *Towards 2020 Science*, a report of 'The 2020 Science Group'. `http://research.microsoft.com/en-us/um/cambridge/projects/towards2020science`, 2005.

[12] Wing J. M. *Computational thinking.* Comm. ACM. Vol. 49 (2006), No. 3, pp. 33-35.

[13] Wing J. M. *Five deep questions in computing.* Comm. ACM. Vol. 51 (2008), No. 1, pp.58-60.

[14] York B. W. *Computational thinking, abstraction and programming: a personal perspective.* `http://web.cecs.pdx.edu/~york/CTshort.pdf`, 2008.