Philosophy & Technology
https://doi.org/10.1007/s13347-020-00411-w

RESEARCH ARTICLE

Computing and Programming in Context—Introduction

Tomas Petricek[1]

Received: 25 March 2020 / Accepted: 14 June 2020/Published online: 09 July 2020

Abstract

In a society where computers have become ubiquitous, it is necessary to develop a broader understanding of the nature of computing and programming, not just from a technical viewpoint but also from a historical and philosophical perspective. Computers and computer programs do not exist in a vacuum. Instead, they are a part of a rich socio-technological context that provides ways for understanding computers and reasoning about programs. This includes not only formal logic, mathematics, sciences, and technology but also cognitive sciences and sociology. The focus of this special issue is on questions that arise when we consider computing and programming in a wider context. In particular, the papers in this special issue explore the interplay between computing or programming and mathematics, formal logic, sciences, technology, and society.

Keywords Philosophy of programming · Formal logic · Simulations · Natural language processing

Mathematics and formal logics—What role have mathematics and formal logics played in the history of computing and programming? What is the nature of the relationship between computer programs as technical computing artifacts and their formal models?
Sciences—Does computing and programming provide qualitatively new methods in sciences such as physics or biology? What can we learn by tracing the interaction between computing and scientific knowledge through the history?
Technology—In what ways have technological innovations enabled developments in computing and programming? What is the nature of the technological artifacts used in computing and how does it differ from other areas of technology?
Society—What are the societal implications of computing and programming? How are developments in computing interlinked with activities of professional

✉ Tomas Petricek
t.petricek@kent.ac.uk

[1] University of Kent, Canterbury, UK

organizations or businesses? How does programming contribute to disciplines outside of a narrow business programming context in areas such as art?

This special issue brings together works exploring computing and programming across their rich formal mathematical, scientific, technological, and social contexts. We are convinced that an interdisciplinary approach is necessary for understanding computing and programming in their multifaceted nature. As such, this issue presents interdisciplinary submissions by researchers coming from a diversity of backgrounds, including historians, philosophers, and computer scientists.

1 Interdisciplinary Approach to Computing and Programming

The special issue is based on the work of the DHST/DLMPST Commission on the History and Philosophy of Computing (www.hapoc.org), which was established in 2013. The Commission aims to create opportunities for collaborations and discussions across disciplinary and methodological boundaries. It engages with all aspects of computing and programming including the philosophy and the history but also its technical and formal aspects. We are convinced that it is only by being embracive and tolerant with respect to different viewpoints, methods, and topics that it is possible to develop a history and philosophy of computing which accounts for both the scientific, social, and technological aspects of the discipline.

The Commission organizes two series of interleaving events—a larger and broader one, the Conference on the History and Philosophy of Computing (HaPoC), and a smaller and a more focused one, the Symposium on the History and Philosophy of Programming (HaPoP). This special issue follows two recent events, the fourth HaPoC conference, held in Brno, Czechia, in October 2017 and the fourth HaPoP symposium, held in Oxford, UK, in April 2018. Several of the papers presented in this special issue are full papers based on talks given at those events.

The interdisciplinary approach advocated by the Commission also inevitably places a burden on the reader of this special issue. The individual papers in this issue follow different approaches and methods. The unifying theme is that each paper combines computing or programming with at least one other discipline. As such, you will find here papers that are thoroughly historical, highly technical, and political. However, following the open and collaborative approach of the Commission, all of the presented papers are written with a broader audience in mind. This special issue presents 7 papers that embody the interdisciplinary approach to studying computing and programming advocated by the Commission on the History and Philosophy of Computing.

2 Computing and Programming Meet Philosophy and Culture

The first two papers consider the history of technical concepts, namely, metasyntactic variables and middleware from broader philosophical and cultural perspectives.

In *Foo, bar, baz...: The metasyntactic variable and the programming language hierarchy*, Brian Lennon follows the history of naming metasyntactic, or deliberately meaningless, variables using the terms “foo,” “bar,” and similar. Lennon points out that

such naming of meaningless variable is an element of a particular programming culture. A variable name, which means nothing to a machine and is not even needed for understanding the purpose of the program code here, has a fundamental cultural meaning. This is substantiated by an investigation looking at historical uses of the term "foo" but also domains which explicitly discourage their use such as the software craftspersonship movement. The paper concludes by drawing an inspiring analogy between the technical notion of "pointer" and a cultural interpretation of pointers—metasyntactic variables like "foo" and "bar" are cultural pointers that link the program to a broader programming culture and history.

In *Middleware's Message: The Financial Technics of Codata*, Michael Castelle traces the history of middleware—essential subsystems for asynchronous messaging that have been, and still remain, a vital component of many software systems not just in the finance domain. On the historical side, the paper follows the development of middleware from the appearance of a "stock ticker," a telegraphic device introduced at the New York Stock Exchange in the 1860s, to the contemporary publish/subscribe programming abstractions. On the technical side, the paper identifies the stock ticker as an instance of "codata," a notion of potentially infinite data streams used in contemporary programming research. Finally, on the philosophical side, the paper explores links between the ticker and the work of early twentieth-century philosophers of synchronous experience, simultaneous sign interpretations, and flows of discrete events (Bergson, Mead and Peirce, Bachelard).

3 Computing and Programming Meet Formal Mathematical Logic

The second two papers of this special issue look at the relationships between research on functional and logic programming and work in formal logic, more specifically, combinatorial logic and proof theory.

In *From Curry to Haskell – Paths to abstraction in programming languages*, Felice Cardone studies conceptual links between notions in Curry's 1920s ideas on combinatorial logic and corresponding notions in 1990s source code written in the functional language Haskell. The paper studies the ways in which Curry's work on logic, motivated by problems in the foundations of mathematics, provides concepts that nowadays serve as the foundations of the functional programming paradigm. Those concepts include formal systems which, like functional programming languages, strive towards abstraction; the use of function application as the only primitive for combining the objects of combinatory systems or, later, program fragments; and the inversion principle of formal systems, which is linked to the functional programming notion of folds.

In *Reciprocal influences between proof theory and logic programming*, Dale Miller provides another example of why we need to look beyond narrow disciplinary boundaries in order to understand important developments in the history of programming. The paper traces a fruitful interaction between mathematical logic and programming. Writing from the perspective of a participant in those developments, Miller introduces important proof-theoretic ideas that found use in research on programming languages and vice versa. In the former direction, sequent calculus, linear logic, and higher-order quantification influenced logic programming and programming language research more

generally. In the latter direction, new ways of structuring proof search in logic programming inspired richer analyses of proofs.

4 Computing and Programming Meet Science and Engineering

Running scientific simulations and performing engineering computations has been an important domain for computing and programming since the era of the first digital computers. The next two papers in the special issue consider the history and philosophy of applied computing in those two domains.

In *A formal framework for computer simulations: surveying the historical record and finding their philosophical roots*, Juan Manuel Durán uses a historical approach to shed new light on the philosophical question of "how do computer simulations explain?". The paper looks at definitions of the notion of computer simulation throughout the history and identifies two different ways of thinking about simulations. According to the first viewpoint, simulations are finding a set of solutions in a mathematical model. According to the second, simulations are descriptions of patterns of behavior. The two viewpoints form a methodological map of computer simulations, but they also allow the author to clarify philosophical assumptions that different approaches to computer simulations entail.

In *Concepts of solution and the finite element method: A philosophical take on variational crimes*, Nicolas Fillion and Robert M. Corless explore a methodological issue with typical real-world use of the finite element method for solving partial differential equations. Computational convenience is often chosen over mathematical soundness, which is known as "variational crime." According to the authors, claiming that "the crime does pay" would be too simplistic. They distinguish two different strands of accuracy used to evaluate inexact solutions and document how applied mathematicians deploy much ingenuity to counteract the damage that could be caused by the committed variational crimes. Although the paper has a specific focus on one concrete "crime" in the context of applied mathematics, we can also see it as an example of a ubiquitous conflict between computing in theory and practice.

5 Computing and Programming Meet Linguistics

Finally, the last paper of this special issue uses a broader philosophical perspective to look at a recent development of deep neural network models in computational linguistics.

In *Why can computers understand natural language? The structuralist image of language behind word embeddings*, Juan Luis Gastaldi explores the remarkable recent success of deep neural network models in the field of Natural Language Processing. The paper attempts to depict the image of language that those new computational models offer to us. Much work on deep neural network models (such as word2vec) has been driven by technological motivations and innovations, but the image of language it offers is not so unfamiliar. Most of the underlying mechanisms can be explained by well-known techniques in computational linguistics, and the paper links the resulting image of language to structuralist roots dating back to Saussure.

6 Concluding Remarks

The contents of this special issue reflect the typical contents of presentations at events organized by the Commission on the History and Philosophy of Computing, including the HaPoC 2017 conference and HaPoP 2018 symposium. At a first glance, it may seem difficult to find a common theme among the individual papers. Indeed, the topics range from epistemology of scientific simulations to cultural references in variable naming! The theme joining together the papers in this special issue is not a single problem or a single topic but a shared approach. All papers presented here study their topics with an open-minded interdisciplinary method. They recognize that we cannot truly understand deep neural networks without considering work done in linguistics or that we cannot talk about software engineering concepts such as middleware without being aware of discussions in the philosophy of time.

The call for papers for this special issue received 14 submissions, out of which 7 were eventually accepted. Many of the 14 submissions were based on earlier presentations at HaPoC 2017, which featured 21 talks over 3 days, and HaPoP 2018, which was a one-day event featuring 10 talks. This special issue would not be possible without the hard work of the organizers of those events, support of the HAPOC Commission, and the many anonymous reviewers of the submitted papers. Last but not least, we are grateful to the editors of the Philosophy & Technology journal, namely, Luciano Floridi and Giuseppe Primiero, whose support throughout the editorial process was indispensable. Some of the work on this special issue has been supported by the ANR project PROGRAMme (ANR-17-CE38-0003-01).