Programming language theory: Thinking the unthinkable

Tomas Petricek
Computer Laboratory
University of Cambridge
tomas@tomasp.net

Our thinking is shaped by basic assumptions that we rarely question. Such assumptions exist at multiple levels. Foucault's episteme grounds knowledge within a particular epoch; Kuhn's research paradigms determine how scientists of a given discipline approach problems and Lakatos' research programmes provide undisputable assumptions followed by a group of scientists.

In this paper, we attempt to uncover some of these hidden assumptions in the area of programming language research. What are some of the hidden assumptions that we never question and that determine how programming languages are designed? And what might the world look like if we based our thinking or scientific method on different basic principles?

1. Incommensurability

Science is often distinguished from other human activities by its progressive nature. It has standards for identifying improvements and methods for accumulating them into an ever improving body of sound knowledge. Such traditional view has been challenged by many philosophers[1]. The most significant blow to the idea of progressive science is *incommensurability*[2] – the idea that two theories do not share a common basis that would allow evaluating them using a common metric. Incommensurable theories do not only have different basic assumptions, but they also ask incompatible questions.

In this paper, I first describe different sources of incommensurability described by philosophers and I review how the incommensurability surfaces in programming language research. Finally, in the most speculative part of the paper, I consider what might theories incommensurable with current main-stream programming language research look like, what questions would they ask and attempt to answer.

1.1 Episteme and human knowledge

Michel Foucault's concept of *episteme*[3] captures an incommensurability at the most fundamental level of human knowledge. An episteme defines the assumptions that make human knowledge possible in a particular epoch. It provides the apparatus for separating what may from what may not be considered as scientific. In other words, it determines claims about which we can say whether they are true.

Foucault gives an example of the incommensurability between the earlier episteme of the Renaissance and the later classical episteme. The former is characterized by signs and resemblances while the latter is characterized by ordering and categorization. Natural historian Comte de Buffon (of the classical episteme) refers to the work of Ulisse Aldrovandi (of the Renaissance episteme):

> *Buffon was to express astonishment at finding in the work of a naturalist like Aldrovandi such an inextricable mixture of exact descriptions, reported quotations, fables without commentary, remarks dealing indifferently with an animal's anatomy, its use in heraldry, its habitat, its mythological values or the uses to which it could be put in medicine or magic.*

As Foucault explains, Aldrovandi's report was not incorrect. It linked things in accordance with the system of signs and similitudes, which was incommensurable with the system of ordering and categorization that was assumed by Buffon due to the classical episteme. This quote gives a vivid example of the effect that grounding in a different episteme gives – Aldrovandi's work attempts to answer questions that Buffon (or modern reader) cannot even conceive.[4]

---

[1] For an extended discussion, see e.g. Niiniluoto (2015)
[2] Two most influential philosophy of science works introducing the idea are Kuhn (2012) and Feyerabend (1993)
[3] Foucault (2001), the quoted example appears on page 43
[4] The classical episteme of de Buffon is incommensurable with our modern thinking, but for other reasons

1.2 Paradigms and research programmes

Foucault's episteme considers the whole of human knowledge and may appear too remote for a discussion about particular field of modern computer science. However, related concepts in philosophy of science capture common assumptions in a given scientific discipline or sub-group of scientists.

According to Kuhn[5], normal science is governed by a single *paradigm*. The paradigm sets standard for legitimate work within the science it governs. A paradigm is not explicitly defined. Instead, it is formed by the methods and assumptions that a scientist learns during her training. When a paradigm becomes insufficient for solving ordinary scientific problems (puzzles), a *paradigm shift* replaces the predominant paradigm with a new one that is *incommensurable* with the old one. As with Foucault's episteme, the new paradigm considers different questions to be worth a scientific enquiry.

At a smaller scale, shared assumptions and methods used by a group of scientists have been captured by Lakatos[6] as *research programmes*. A research programme recognizes that, even in regular science, some laws and principles are more basic than others. Faced with an experimental failure, a scientist never blames such *hard core* assumptions, but instead addresses the issue by modifying some of the additional assumptions provided by the *protective belt* of the theory. Due to the different hard core assumptions, the work arising from different research programmes is to some degree incommensurable.

2. Episteme and paradigms in programming

Do episteme, paradigms and research programmes affect how programming language research is done? In this section, I consider two examples that answer the question in affirmative.

2.1 Mathematization of computer science

Mathematical methods are a foundational part of modern computer science. However, the history of the discipline suggests that this was not a logical necessity. Early programmers were often seen as anything from clerical workers to chess players and artists. In fact, early study noted that majoring in mathematics was not found to be significantly related to performance as a programmer.[7] The focus on mathematics was, however, a good tactical move for early academic computer science:

> *The rise of theoretical computer science was anything but inevitable. (…) Advocates of theoretical computer science pursued a strategy that served them well within the university, but increasingly alienated them from their colleagues in the industry.*[8]

Mathematization of computer science established it as a legitimate academic discipline and differentiated it from industrial computer engineering. An essential part of the development was the concept of *algorithm*, which provided aspiring scientists with a practical agenda for advancing the discipline. Using the terminology of Kuhn, computer science became a *normal science* that is preoccupied with puzzle solving activity. The newly founded research *paradigm* determines which questions are scientific (various questions about algorithms) and how answers should be sought (through formal methods).

It is hard to imagine computer science where algorithm is not a foundational concept, but explaining that as a necessity would be misleading. Rather, a historical coincidence made algorithm a core part of our paradigm. It is even harder to question the idea that mathematics can find relevant answers to the questions posed by programming. This is because mathematization has become a part of our modern episteme. But how might a computer science look when our episteme or research paradigms change?

2.2 The Algol research programme

Paradigm shifts are rare and the change of episteme even more so. At a smaller scale, much of the modern academic programming language research has been influenced by the Algol research programme. The goal of the programme is to utilize the resources of logic to increase the confidence in the correctness of programs.[9]

---

[5] Kuhn (2012)
[6] Lakatos (1978)
[7] Ensmenger (2012), p129
[8] ibid., p117
[9] ibid., p257

The Algol research programme defines the hard core, together with a sufficiently open ended agenda. Indeed, much of the theoretical programming language research aims to prove programs correct through the use of formal logic. The methods differ, but the hidden assumption that formal proof provides the correct methodology is widely shared. Any experimental failures (such as the fact that proving programs correct is still difficult after 50 years of the existence of the Algol research programme) are attributed to protective belt – we do not yet have sufficiently powerful formal methods, the problem is not properly formally specified and programmers in the industry are not using the right tools.

3. Thinking the unthinkable

I outlined how programming and science more generally are affected by assumptions that are implicit in research programmes, scientific paradigms and also the episteme of the current period. I considered concrete examples from programming language research to illustrate that those are not just abstract philosophical concepts.

As suggested by Ensmenger[10], the establishment of theoretical computer science rooted in mathematics was not an inevitable development and it is conceivable that computer science would evolve differently, building on principles other than algorithms and formal logic. Similarly, in programming language research, the predominant Algol programme is not the only one. A largely incommensurable research programme was defined by Smalltalk where "programming was not thought of as the task of constructing a linguistic entity, but rather as a process of working interactively with the semantic representation of the program, using text simply as one possible interface."[11]

The most interesting aspect about research programmes, scientific paradigms and episteme is that the competing programmes or paradigms and episteme that replace the old ones are incommensurable with the old ones. This means that they do not share the same assumptions, goals and ways of thinking. The rest of the essay speculates on what might programming research arising from a different paradigm or episteme look like.

3.1 Taxonomies of programming ideas

Theoretical computer scientists attempt to extract mathematical essence of programming languages and study its formal properties.[12] Now consider an episteme that instead aims to explore the design space and build a *taxonomy* of objects that occupy the space.[13] It considers the entities as they are, rather than trying to extract their mathematical essence. What would be the consequence of such way of thinking that attempts to relate and organize programming ideas in taxonomies, rather than abstracting?

In programming language research, many novel ideas that defy mathematization are left out because they are too "messy" to be considered through the predominant formal perspective. If the episteme made us seek relationships, those would all became within the realm of computer science. For example, we would be able to discover similarities between live coded music and formula editing in Excel[14] as both of those represent a form of programmer interaction with immediate observable feedback.

Science should not merely observe, but also "twist the lion's tail" and conduct experiments to probe the properties of the nature[15]. If our focus is on building taxonomies, the nature of relevant experiments will also differ. Rather than measuring properties of simple models, experiments designed to reveal relationships need to highlight interesting aspects of a behaviour in its full complexity. They need to reproducibly demonstrate relationships that are not immediately obvious, similarly to how thought-experiments in sciences and philosophy highlight an intriguing aspect of a theory. An interesting format that appeared recently is the presentation of programming research in the form of screencasts.[16]

---

[10] Ensmenger (2012)

[11] Priestley (2011), p294

[12] An occasional attempt to treat programming differently has been made, for example, by Noble (2002)

[13] This way of thinking is similar to Foucault's classical episteme that focused on the construction of taxonomies

[14] Thanks to Sam Aaron (author of live-coding environment Sonic Pi) for numerous inspiring discussions on this topic

[15] The quote has been attributed by many philosophers of science to Francis Bacon, founder of modern scientific method

[16] This is the format used by the Future Programming workshop (http://www.future-programming.org)

Another consequence of the focus on taxonomies in the classical episteme was the creation of museums, which present the studied objects neatly organized according to the taxonomy. A computer scientist of such alternative episteme might follow similar methods. Rather than finding mathematical abstractions and presenting abstract mathematical structures, she would build (online and interactive?) museums to present typical specimen as they appear in interesting situations in the real-world.[17]

3.1 Crossing the vertical gaps with metaphors

Does modern computer science have something to gain from the Renaissance episteme centred around signs and resemblances? Work on programming languages is done at three layers. At the top layer, language or library designs follow some intuitive ideas, which are turned into an actual program (middle layer). Formal reasoning is done about a simplified mathematical model, which is the lower layer. Programming research operates horizontally – relating different formal models or various implementations – but does not easily cross between the layers. Theoretical work at the bottom is rarely linked with the informal top layer.

Signs and resemblances provide the missing link between layers. In object-oriented design, an object is a *metaphor* for an object in the real-world,[18] but metaphors can be found in many areas of programming. We use them to conceive an idea and use our intuition at the higher level to guide design at lower levels. Since our episteme does not consider such resemblances important, we then hide them (for brevity, or out of disinterest) from our published narrative. Metaphors often remain present only through naming. In John von Neumann's First Draft of Report on EDSAC[19] (which described modern computer architecture) individual units are called "organs" suggesting a biological metaphor for the system structure. Even more obvious metaphors are hidden in the plain sight. For example, when and why did we start calling programming languages "languages"?

If resemblances and metaphors played fundamental role in our scientific thinking, we would not just gain interesting insights from them, but we would also ask different questions (which appear secondary or unscientific when considered through the perspective of our current way of thinking). What research agenda would computer scientist (or a programming literary critic) of an episteme centred around metaphors ask? Looking at the problem of concurrent programming as an example, one might try to design programming language models for concurrency by studying how simultaneity is expressed in different forms in narrative.[20] What are the postmodern alternatives to threads where concurrent processes occurring in the same temporal interval are described in successive textual parts of the narrative?[21]

4 Conclusions

Overall way of thinking that is captured by episteme or paradigms has been changing throughout the history and we can expect it to continue changing. Ideas grounded in different episteme or paradigms are often incommensurable, meaning that they have different basic assumptions and consider different questions as scientific. Just like Buffon was astonished when reading the work of Aldrovandi, we may wonder which of our current scientific achievements will appear as exact anatomical descriptions and which will appear as fables, and magic to the thinkers of the future.

Were we to think about programming in terms of taxonomies, much of the present work that explores interesting aspects of, say, programming language design space will still be relevant. The readers might be astonished why we focus on irrelevant technical details rather than trying to present the most unique interesting aspects of the work in their full richness. Were we to think about programming in terms of similarities and metaphors, formal mathematical models will become just one of many forms of metaphors available when understanding programs, but future thinkers might be astonished by our attempts to find more and more remote abstractions that gradually lose more and more of the similarity with the original idea and become merely a work of art or fiction.

---

[17] The Design Patterns book by Gamma et al. (1994) can perhaps be seen as an example of this approach

[18] Noble (2002)

[19] von Neumann (1945)

[20] Sedlacek (2011) looks at economics (mathematicised similarly to programming) through a perspective based on myths

[21] This unsophisticated style of narrative is mockingly referred to as "Meanwhile, back on the ranch." See Margolin (2016)

5. References

Niiniluoto, I. (2015) Scientific Progress. The Stanford Encyclopedia of Philosophy, Edward N. Zalta (ed.), Available at http://plato.stanford.edu/archives/sum2015/entries/scientific-progress

Kuhn, T. S. (2012). The structure of scientific revolutions. University of Chicago Press

Feyerabend, P. (1993). Against method. Verso

Foucault, M. (2001). The Order of Things: An Archaeology of the Human Sciences. Routledge

Lakatos, I. (1978). The methodology of scientific research programmes. Philosophical Papers Volume 1. Cambridge University Press

Ensmenger, N. L. (2012) The computer boys take over: Computers, programmers, and the politics of technical expertise. MIT Press

Priestley, M. (2011). A science of operations: machines, logic and the invention of programming. Springer Science & Business Media

Noble, J., Biddle, R. (2002) Notes on postmodern programming. In Proceedings of Onward!

Noble, J., Biddle, R., Tempero, E. (2002) Metaphor and metonymy in object-oriented design patterns. In Australian Computer Science Communications, vol 24, issue 1

Gamma, E., Vlissides, J., Johnson, R., Helm, R. (1994) Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley

von Neumann, J. (1945). First Draft of a Report on the EDVAC. University of Pennsylvania

Margolin, U. (2016). Simultaneity in Narrative. In: Hühn, P. et al. (eds.): The living handbook of narratology. Hamburg: Hamburg University Press.

Sedlacek, T. (2011) Economics of Good and Evil: The Quest for Economic Meaning from Gilgamesh to Wall Street. Oxford University Press