Trends in Cognitive Sciences

CellPress
REVIEWS

Science & Society

The Language of Programming: A Cognitive Perspective

Evelina Fedorenko,[1,2,3,*,@] Anna Ivanova,[1] Riva Dhamala,[4] and Marina Umaschi Bers[4,*,@]

Computer programming is becoming essential across fields. Traditionally grouped with science, technology, engineering, and mathematics (STEM) disciplines, programming also bears parallels to natural languages. These parallels may translate into overlapping processing mechanisms. Investigating the cognitive basis of programming is important for understanding the human mind and could transform education practices.

The Growing Importance of Computer Programming

In the automated economy, computer programming is becoming an essential skill across diverse fields and disciplines. As a result, countries across the world are exploring the inclusion of computer science (CS) as mandatory in the school curriculum. For example, the US government launched the 'Computer Science for All' initiative in 2016 to introduce programming at all educational levels. Similar initiatives are taking place across Europe, Asia, and South America.

The growing importance of CS education underscores the urgency in characterizing the cognitive mechanisms and the corresponding brain circuits that support the acquisition and use of computer programming skills (Box 1). This basic knowledge is needed to guide the design of curricula, assessments, and educational policies regarding when and how to introduce CS in schools and to inform the implementation of teaching strategies and disciplinary integration.

Furthermore, an understanding of the cognitive and neural basis of programming can contribute to the general enterprise of deciphering the architecture of the human mind. Computer programming is a cognitive invention, like arithmetic and writing. How are such emergent skills acquired? Presumably, they rely on phylogenetically older mechanisms, many of which we share with other animals, but which mechanisms? And how do these mechanisms and new domains of knowledge interact with the evolutionarily older and ontogenetically earlier-emerging ones?

Programming as Problem Solving

Traditionally, many researchers have construed programming in terms of problem solving, dividing it into distinct steps: problem comprehension, design, coding, and debugging/maintenance [1]. As a result, when describing the cognitive underpinnings of programming, researchers have often focused on the early stages of program planning and the ability to break down a problem into discrete units (later dubbed 'computational thinking' [2]). Studies that have probed the process of coding itself have often resorted to evaluating overall cognitive load [3]. Thus, empirical research has lagged behind in exploring the relationship between mechanisms that underlie programming and other cognitive skills, in particular language ability. Despite an abundance of metaphoric descriptions linking computer and natural languages, such as the use of the terms 'syntax' and 'semantics' [4], the problem-solving approach has continued to dominate the discourse in the field of CS education and sometimes eclipsed research exploring other cognitive mechanisms potentially involved (see the supplemental information online for a more detailed overview).

Beyond STEM: An Alternative Construal of CS

Despite the lack of rich and detailed characterization of the cognitive bases of computer programming, educators have long made assumptions about the relationship between programming and other cognitive skills. These assumptions have shaped the treatment of CS in schools across the world as mathematically/problem-solving oriented, and, when integrated in the curricula, CS has been grouped with STEM disciplines [5]. However, some have argued for an alternative construal of programming – an approach that has become known as 'coding as literacy' [6]. The key idea is this: when you learn a programming language, you acquire a symbolic system that can be used to creatively express yourself and communicate with others. The process of teaching programming can therefore be informed by pedagogies for developing linguistic fluency.

The term 'programming languages' already implies parallels to natural language. However, to rigorously evaluate the nature and extent of potential overlap in the cognitive and neural mechanisms that support computer versus natural language processing, it is critical to delineate the core components of each process and formulate specific hypotheses about the representations and computations that underlie them. Here, we propose a framework for generating such hypotheses.

With respect to knowledge representations, both computer and natural languages rely on a set of 'building blocks' (words and phrases in natural language, functions and variables in computer languages) and a set of constraints for how these building blocks can combine to create new, complex meanings. Studies dating back to the 1970s have noted this parallel, as evidenced by the occasional reference to the semantics and syntax of programming languages [4], but few have investigated this distinction experimentally (see the supplemental information online). Whereas the technical meanings of these terms in linguistics and CS differ, their usage highlights that both natural and programming languages rely on meaningful and structured representations.

Box 1. Why Now? The Urgency of Understanding the Cognitive Underpinnings of Computer Programming

Given the growing demand for programming skills, educators have been increasing efforts to make CS classes available to students. The question of whether CS should be grouped with STEM or with languages has sparked countless debates, with some citing Dijkstra who claimed that 'mastery of one's native tongue' is key to competent programming [14] and Papert's early vision of 'learning to program as a second language' [15] and others pointing to the decades-long tradition of viewing programming as principled problem solving. Lawmakers have also weighed in on the issue. In 33 US states, CS fulfills a mathematics or science requirement, with Texas and Oklahoma providing an option to count it as a foreign language (https://code.org/promote). At the federal level, legislators have proposed an initiative to award grants to schools that count CS toward either a mathematics/science or a foreign language requirement.

There is a marked lack of scientific research that would support any of those initiatives. Although programming may, to some extent, recruit both STEM and language skills, no studies of programming have so far examined the exact division of labor between these cognitive domains. As new educational policies are introduced, understanding the cognitive underpinnings of programming can help to guide those decisions.

With respect to computations, a multistep processing pipeline appears to underlie both the comprehension and the generation of linguistic/code units (Figure 1). In comprehension, we start with perceptual input and are trying to infer the intended meaning of an utterance or to decipher what a piece of code is trying to achieve. In doing so, we initially engage in some basic perceptual processing (auditory/visual in natural languages and typically visual in computer languages) and then attempt to recognize the building blocks and the relationships among them. For longer narratives and extended pieces of code, we need to not only understand each utterance/line but also to infer an overall high-order structure of the text/program.

In generating linguistic utterances or code, we start with an idea. This idea can be a simple one and require a single sentence or line of code or it can be highly complex and require a whole extended narrative or multipart program. For simpler ideas or subcomponents of complex ideas, we need to figure out the specific building blocks to use and to organize them in a particular way to express the target idea. For more complex ideas, we first need to determine the overall structure of the narrative or program. Once we have a specific plan for what we want to say, or what a piece of code would look like, we engage in actual motor implementation by saying/writing an utterance or typing up code statements. It is worth noting, however, that in generating both linguistic texts and computer code, top-down planning may be supplemented with more bottom-up strategies where certain fragments of the text/code are produced first or borrowed from previously generated text/code, and then the overall structure is built around those.

During these comprehension/generation processes, we also engage in other, plausibly similar, mental computations. For example, we can recognize errors – others' or our own – and figure out how to fix them [1]. When processing sequences of words and commands, we plausibly engage in predictive processing: as we get more input, we construct an increasingly richer representation of the unfolding idea, which in turn constrains what might come next. Further during the generation of utterances or code, creative thinking comes into play, affecting the very nature of the ideas one is trying to express as well as how those ideas are converted into sentences/code. Finally, we may need to consider the intent of the producer or the state of mind of our target audience – abilities that draw on our mentalizing (Theory of Mind) capacities (although mentalizing about the computer itself might be maladaptive; see the supplemental information online).

It is also worth noting that the vast majority of programming languages directly rely on programmers' knowledge of natural languages (specifically, English). Keywords, variable names, function names, and application programming interfaces follow naming conventions that indicate their function; it has been shown that 'unintuitive' naming increases cognitive load [7] and hinders program comprehension [8]. The importance of natural language semantics is further highlighted by the fact that non-native English speakers often struggle to learn English-based programming languages [9]. Further, computer code is usually accompanied by comments and, for larger pieces of software, documentation, which serve to scaffold program comprehension. Thus, the process of working with code necessarily involves tight integration of computer and natural language knowledge.

The machine learning community has already begun to exploit structural similarities between code and natural language by applying natural language processing techniques to analyze code [10]. Developmental psychology researchers have also begun to explore those parallels. A case study [11] demonstrated that knowing a programming language can facilitate the acquisition of reading ability. Pilot studies with preschoolers and kindergarteners have also shown that programming can facilitate language processing, as young children's sequencing abilities significantly improved after they received coding interventions [12]. Neuroimaging studies of programming, although in their infancy, hint at potential overlap between language- and code-processing brain regions [13]. Such findings, along with the theoretical framework presented above, call for direct investigations of cognitive and neural overlap between language and code processing.

Concluding Remarks

The growing importance of computer programming underscores a need to conduct rigorous research probing the cognitive

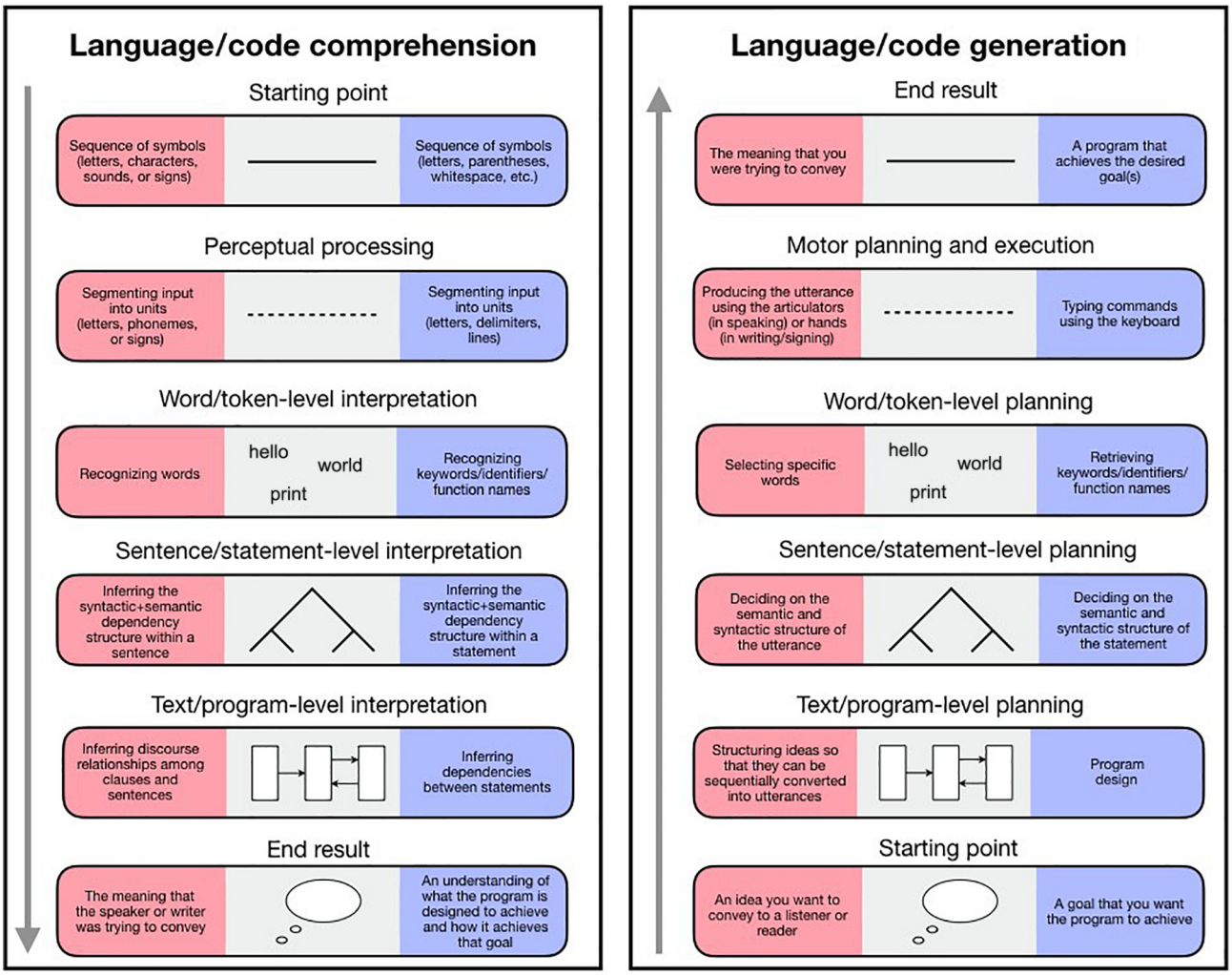

Figure 1. Hypothesized Parallels between Natural Language (Red) and Computer Programming (Blue) at Different Processing Stages. Both sets of cognitive processes rely on the combinatorial nature of their inputs and outputs, which comprise phonemes/letters that make up words/identifiers, which combine into sentences/statements, giving rise to paragraphs and functions, and finally yield texts/utterances/programs (cf. the supplemental information online for possible differences).

architecture that underlies programming abilities. We have highlighted potential parallels between programming and natural languages. Although the comparison is not perfect and some mental computations are likely to differ (see the supplemental information online), future empirical studies should consider the hypothesis that programming draws on some of the same resources as natural language processing, in addition to the traditional proposal whereby programming shares computations with mathematics, logic, and problem solving. If this hypothesis finds empirical support, we need to reconceptualize the way CS is taught, especially in early childhood, when children are learning to read and write. CS learners might also benefit from techniques employed in foreign language classrooms, such as constant exposure to the language and learning by doing. Making progress in deciphering the cognitive and neural bases of computer programming may therefore yield fundamental insights about how to optimally design curricula, policy, and educational interventions, as well as new programming languages for children that might draw on pictorial and not only textual interfaces [15].

Author Contributions

E.F. and M.U.B. developed the idea for the manuscript. E.F., A.I., and M.U.B. wrote the manuscript, with critical input from R.D. A.I. created the figure.

Acknowledgments

This work was supported by an NSF EAGER award (FAIN 1744809, 'The cognitive and neural mechanisms of computer programming in young children: storytelling or solving puzzles?') to M.U.B. and E.F. We also thank three anonymous reviewers for their helpful comments and suggestions.

Supplemental Information

Supplemental information associated with this article can be found online at https://doi.org/10.1016/j.tics.2019.04.010.

[1]Brain and Cognitive Sciences Department, Massachusetts Institute of Technology, Cambridge, MA, USA
[2]McGovern Insitute for Brain Research, Massachusetts Institute of Technology, Cambridge, MA, USA
[3]Psychiatry Department, Massachusetts General Hospital, Boston, MA, USA
[4]Eliot-Pearson Deparmtent of Child Study and Human Development, Tufts University, Medford, MA, USA

*Correspondence:
evelina9@mit.edu (E. Fedorenko) and Marina.Bers@tufts.edu (M.U. Bers).
@Twitter: @ev_fedorenko (E. Fedorenko) and @marinabers (M.U. Bers).

https://doi.org/10.1016/j.tics.2019.04.010

References

1. Dalbey, J. and Linn, M.C. (1985) The demands and requirements of computer programming: a literature review. *J. Educ. Comput. Res.* 1, 253–274
2. Wing, J.M. (2006) Computational thinking. *Commun. ACM* 49, 33–35
3. Nakagawa, T. *et al.* (2014) Quantifying programmers' mental workload during program comprehension based on cerebral blood flow measurement: a controlled experiment. In *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE2014)*, pp. 448–451, ACM
4. Shneiderman, B. and Mayer, R.E. (1975) Towards a Cognitive Model of Programmer Behavior, Indiana University
5. Guzdial, M. and Morrison, B. (2016) Growing computer science education into a STEM education discipline. *Commun. ACM* 59, 31–33
6. Bers, M.U. (2019) Coding as another language: why computer science in early childhood should not be STEM. In *Exploring Key Issues in Early Childhood and Technology: Evolving Perspectives and Innovative Approaches* (Donohue, C., ed.), Routledge
7. Fakhoury, S. *et al.* (2018) The effect of poor source code lexicon and readability on developers' cognitive load. In *Proceedings of the International Conference on Program Comprehension (ICPC)*, pp. 286–296, IEEE
8. Lawrie, D. *et al.* (2006) What's in a name? A study of identifiers. In *Proceedings of the International Conference on Program Comprehension (ICPC)*, pp. 3–12, IEEE
9. Guo, P.J. (2018) Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pp. 396, ACM
10. Allamanis, M. *et al.* (2018) A survey of machine learning for big code and naturalness. *ACM Comput. Surv.* 51, 81
11. Peppler, K.A. and Warschauer, M. (2011) Uncovering literacies, disrupting stereotypes: examining the (dis)abilities of a child learning to computer program and read. *Int. J. Learn. Media* 3, 15–41
12. Kazakoff, E.R. and Bers, M.U. (2014) Put your robot in, put your robot out: sequencing through programming robots in early childhood. *J. Educ. Comput. Res.* 50, 553–573
13. Siegmund, J. *et al.* (2014) Understanding source code with functional magnetic resonance imaging. In *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE2014)*, pp. 378–389, ACM
14. Dijkstra, E.W. (1982) How do we tell truths that might hurt? In *Selected Writings on Computing: A Personal Perspective*, pp. 129–131, Springer
15. Papert, S. (1980) Mindstorms: Children, Computers and Powerful Ideas, Basic Books