The *Communications* Web site, <http://cacm.acm.org>, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

Follow us on Twitter at <http://twitter.com/blogCACM>

DOI:10.1145/3311717

<http://cacm.acm.org/blogs/blog-cacm>

Pondering Variables and Direct Instruction

Robin K. Hill considers the nature of variables, while Mark Guzdial reflects on renewed interest in the “direct instruction model.”

Robin K. Hill
What Is a Variable?
<http://bit.ly/2D9XQri>
January 31, 2019

What is a variable?—the name of a value. What is an attribute?—the name of a property. These are the working definitions that most of us find adequate in daily life and daily computer science. To consider whether there is more to it is to consider an ontology, the ontology of the variable. Is there a good comprehensive definition of “variable” for students and laypersons?

First, however, let's address the subject that arises prominently where variables meet worldly ontology—the professional design of an ontology for some real thing, in some domain, driven by a commercial need to capture some enterprise in a database. The question is, “What do we need to keep track of?”

To answer that question—there's money in it—requires extensive knowledge of some real-world arena of activity, such as sewing, or collateralized debt obligations, or military deployments, or Bordeaux wines,⁴ along with

the skills to represent that knowledge in some data language. Working under the entity-relationship model, an expert would elicit and record the entities, attributes, and relationships that need to be exposed in the domain.

The principles of working ontology is a subject taken up by several philosophers of computing. The goal is often knowledge representation and sharing, and the objective is often organizational. Barry Smith and fellow researchers combine the abstract or formal with the domain-specific.^{6,7} Basic Formal Ontology provides templates and tools as in engineering, and addresses philosophical questions arising from those practices.¹

That's not what we're talking about. The question of interest here—there's no money in it—requires probing into the nature of that workhorse abstraction, the variable. We look not to the fine-grained level of a “distinction that makes a difference” (attributed to Donald McKay³), but to the level of identifiers with associated values. Quine says that “to be is to be the value of a bounded variable,” an intriguing notion, but not the informal explication we seek. We want to articulate—toward a com-

prehensive definition—the ontology of a variable as it is used in computer programming. We mutter, “What's the variable for the tax rate?” or “That variable has a misleading name” or “We don't need a variable for month, because that's a function of the date.” The ontology of something that is used so relentlessly in computing must have something to tell us about computing.

Let's consider several questions that may provide direction. (In fact, questions are as far as we go here.)

► If a variable is the name of a value, what about a constant? The distinction is easy to explain in programming, but not so easy outside of it. Is `const` a special case of `var`, where some extra property, “mutability,” attaches to the `var` object? And what about the *attribute* of database design; what is that? Does `attr` break down into `var` and `const`? William Kent raised this question of the ontology of the attribute in 1978, noted its peskiness, and concluded there is really no difference between an attribute and a relationship.²

► What is the difference between a variable and a value? Is the variable a thing or some property of a thing? To say we don't need a variable for the

month is to say the value (but not the variable) can be derived elsewhere. Because variables “go away” when values are substituted, a common view of a variable is as a “placeholder.” What is it holding a place for? For a predicate? For a proposition? For an entity? In other words, what is the shape or category of a variable? Is it an attribute of something, a whole statement about something, or the something itself? Again, in a program (or in its formalization in denotational semantics), this is clear; the difference between variable and value is embedded firmly in the process.

► What about the multiple instantiations, the fact that all occurrences of a variable are bound to the same value? To say “that variable has a misleading name” is to condemn several substrings of code. How would we include that aspect (also straightforward in a formal semantics) in a comprehensive definition?

► Is a variable something we know or something we don’t know? An algebraic definition of a variable is as an “unknown.” Yet we know a lot about a variable x that appears in an equation such as $f(x) = ax^2 + bx + c$. We know that it is an object suitable for such a setting, that is, a number, and that it belongs right there, and that its value is constrained by the given relations to other values. What’s unknown is which number. We always “know” the variable in the rather weak sense that we grasp its need and purpose. In a program, a variable’s value is something that we don’t know in the source code, the human product. In the execution, as soon as it’s bound, the value is “known” in some sense, but to what or whom?

The art of definition has a philosophy of its own.⁵ Experts in that realm would not be surprised to encounter difficulties, but computer scientists might be surprised at the subtle and complex issues that come from a mundane and familiar idea.

References

1. Buffalo Ontology Site, State University of New York at Buffalo <http://ontology.buffalo.edu/>
2. Kent, W. *Data and Reality*, North-Holland Publishing Company, Chapter 5.0, Attributes.
3. McKay, D. *Information, Mechanism and Meaning*, MIT Press, 1969
4. Noy, N.F., and McGuinness, D.L. *Ontology Development 101: A Guide to Creating Your First Ontology* <https://stanford.io/2HTnFLl>
5. Gupta, A. Definitions, *The Stanford Encyclopedia of*

Philosophy, Edward N. Zalta, editor.

<https://stanford.io/2D5Qhls>

6. Smith, B. Beyond concepts: ontology as reality representation. In *Proceedings of the Third International Conference on Formal Ontology in Information Systems (FOIS 2004)*, IOS Press, pp. 73–84.
7. Smith, B. Ontology (science). Formal Ontology in Information Systems, Eschenbach, C. and Grüninger, M., eds. In *Proceedings of the Fifth International Conference on Formal Ontology in Information Systems (FOIS 2008)*, IOS Press, pp. 21–35. <http://bit.ly/2TBynOB>, DOI:10.3233/978-1-58603-923-3-21

Mark Guzdial Direct Instruction Is Better Than Discovery, But What Should We Be Directly Instructing?

<http://bit.ly/2ARuUTA>

November 7, 2018

I recently discovered Felienne Hermans’ blog. In her latest post, she talks about the research around direct instruction and how it relates to programming. The research evidence is growing that students learn better through direct instruction rather than through a discovery-based method, where we expect students to figure things out for themselves.

Direct instruction is hot! Whereas in the 1990s we heard a lot about discovering learning, we are now slowly seeing a renewed interest in the ‘direct instruction model’ in the Netherlands. Both in language and in mathematics there is a new interest in rote practice of knowledge (“stampen,” <http://bit.ly/2D5pRAa>). As that is not a surprise, since research keeps showing that direct instruction—explanation followed by a lot of focused practice—works well (<https://nyti.ms/2HQbd5n>). It not only works well, it also equalizes: it does not matter what knowledge children already have (received at home), everyone has equal chances to acquire the basic knowledge. That is why research also shows that direct instruction works especially well for weaker pupils (<http://bit.ly/1BAsEOh>).

Her point resonates strongly with me. I argued here in Blog@CACM last year (<http://bit.ly/2RzDDQR>) that we should reduce the amount that we tell students to just “figure out” in CS classes. We should teach students directly (Hermans’ point), and reduce (at least in the first classes) the amount of design and problem-solving we ask students to do.

But *what* should we be teaching directly? The obvious answer is “the programming language.” But there’s

a good bit of evidence that students do not learn the syntax and semantics of programming directly. Kelly Rivers had a paper at ICER 2016 (<http://bit.ly/2BhtuDa>) that I found fascinating. She studied how students learned programming structures, and found that they *didn’t*. They actually made more errors with FOR loops over the course of the semester. Of course, the students were learning, but you can’t measure their learning in terms of syntax and semantics of programming. They’re learning something else.

Elliot Soloway and Jim Spohrer argued back in the 1980s that students don’t learn programming in terms of syntax and semantics (see the CHI 1986 paper at <http://bit.ly/1BAsEOh>). Rather, students were learning *plans*, useful chunks of code. Within those chunks were programming statements, but students were learning that *set* of statements, not the individual statements. Maybe we should be teaching those plans?

John Sweller argues we should be teaching programming with lots of worked examples. He argues that too much emphasis on problem-solving leads to increased cognitive load, which interferes with learning. Worked examples are completely worked out programs that students study, typically including questions that students answer about the program. The worked-example effect (<http://bit.ly/2MPGjZJ>) describes how worked examples lead to better learning than more problem-solving. For Sweller, worked examples are part of effective direct instruction for programming.

I am a follower of the research Hermans is citing in her blog post, and agree that direct instruction is better than discovery learning for introductory courses. We are still figuring out what direct instruction means in learning to program. I do not think it is about programming languages. I expect that it is more about plans, and that the methods should involve worked examples.

Robin K. Hill is a lecturer in the Department of Computer Science and an affiliate of both the Department of Philosophy and Religious Studies and the Wyoming Institute for Humanities Research at the University of Wyoming. **Mark Guzdial** is a professor in the Computer Science & Engineering Division of the Engineering Education Research program at the University of Michigan.

© 2019 ACM 0001-0782/19/4 \$15.00