What Will Our Great Grand-Children Be Learning About Computing?

David S. Touretzky, Computer Science Department, CMU

There is a growing movement today to integrate computing instruction into primary school curricula. The United Kingdom [2] and the Republic of Estonia [4] have already revised their national education standards to mandate programming instruction for all students, and it is likely that over the next few years many US states will follow suit, as 25 states already mandate computer science as an elective at the high school level [1, 7]. But exactly what students should be learning, and how they should learn it, is a contentious issue.

In the U.S., the Common Core standards for mathematics [6] define concepts and skills that every child should possess at each grade level. What will these look like for computing in 50 years?

First we should ask what computing will look like in 2065. Voice, gesture, and natural language input will dominate our casual interactions with technology. Today's smartphones already handle spoken language queries well enough to be useful. But formalism remains the most effective way to describe complex relationships, so some type of formal computational thinking will surely be taught in school as part of the mathematics curriculum.

Does this mean that our great grandchildren will be learning Scratch [8] or Alice [3], two currently popular languages for introducing kids to programming? Probably not. While their drag-and-drop interfaces eliminate syntax errors, the semantics of these languages aren't that different from Java or Python. They do little to shield students from the tedium of programming with primitives that are too low level. I'll sketch what a higher level approach to computing might look like shortly.

The essence of computational thinking is *lawful manipulation* of *structured data*. Students can be introduced to these concepts early in the K-6 computing curriculum through specific examples. I would argue that it is state machines that are the fundamental mechanism of lawful behavior that we will be teaching. State machines are ubiquitous in computer science, appearing in such diverse areas as automata theory, digital logic design, network protocols, and robot programming. Understanding the notion of *state* and learning to think in terms of state machines forever alters the way one perceives digital technology, from microwave ovens to smartphones.

Conventional procedural languages do not provide explicit support for state machines. Programmers must implement state machine logic using some combination of variables, conditionals, and the program counter. Bur some robot programming formalisms are organized around state machines. These include the NXT-G language used in LEGO Mindstorms, Choregraphe for the Nao, SMACH for ROS (the Robot Operating System), and my own Tekkotsu framework. Microsoft's Kodu Game Lab [5], while presented as an environment for novice game developers, is actually a behavior-based robotics programming environment and explicitly represents states as "pages" of WHEN-DO rules. State machines are the first half of the answer to how our great grandchildren will be introduced to computational thinking.

The second half is structured data. Again there is a fundamental concept: graphs, but graphs are too abstract to start with. There is however a special case that is especially compelling. Trees are the key structure that children should be taught first, because they are encountered in so many places in our daily lives. A book's table of contents is a tree; an organizational chart is a tree; a part-whole hierarchy is a tree; and so is the menu system in any device with a graphical user interface. So while today's languages for children introduce structured data through arrays or lists, in the future I think children will start with trees, encounter lists as a special case, and then move on to more general graph structures. Kids' programming languages that support this approach

with good visual interfaces for tree manipulation have yet to be designed.

Instruction in computational thinking involves more than just programming, just as mathematics instruction involves more than using a calculator or Mathematica. Learning to reason about state machines and data structures in a computing class (or about functions and equations in a math class) is why children will still need to go to school. But their interactions with technology at a young age outside the classroom mean they won't have to be taught some things. Just as today's children don't require formal instruction in how to use a mouse or a game controller, our great grandchildren will learn a lot about computing by watching their parents and older siblings, by using household appliances, and by playing with intelligent robot toys.

What I call the "fourth tier" of robot toys does not exist today, but is clearly on the horizon. Tier 1 robot toys offer direct control of actuators, as in a radio controlled car or plane. Tier 2 toys have more degrees of freedom, making them too complex to control at the actuator level, so they offer a repertoire of predefined behaviors that can be activated individually or strung together into linear sequences. Wowee's Robosapien, a 13.5 inch tall, roughly humanoid robot with canned gestures triggered by remote control, is a popular example. Tier 3 toys are user programmable, but at a very low level by direct control of individual actuators. Sensing is limited to bump switches or light detection. The LEGO Mindstorms and VEX robotics kits are examples of tier 3 technology.

In much less than 50 years we will have self-driving cars, intelligent information assistants, and tier 4 robot toys. These robots will have cameras and computer vision algorithms that allow them to perceive the world and maintain a map of their environment. They will have kinematics and path planning software so they can grasp and manipulate objects. And they will be programmable by children using high level primitives for perception and action, similar to what Kodu provides today. Think of the technology in today's best graduate robotics labs, but with intuitive interfaces, shrunk to platforms suitable for children to play with, and made affordable through mass production. As our great grandchildren grow up surrounded by computing technology, they will play with toys that implement lawful behaviors and construct complex representations of the world. And they will take their next steps toward becoming adept computational thinkers.

References

[1] Armitage, A. (2015) Arkansas gov. directs high schools to offer computer science classes. *Education Week* Digital Edition, February 26, 2015. Accessed online at <http://blogs.edweek.org/edweek/DigitalEducation/2015/02/arkansas_gov_directs_high_scho.html>

[2] Berry, M. Computing in the national curriculum: A guide for primary teachers. *Computing At School*, 2013. Available online at <www.computingatschool.org.uk/primary>, accessed June 23, 2015.

[3] Dann, W.P., Cooper, S., and Pausch, R. (2011) *Learning to Program with Alice*, 3rd edition. Upper Saddle River, NJ: Prentice Hall.

[4] Olson, P. Why Estonia has started teaching its first-graders to code. *Forbes*, September 6, 2012. Retrieved March 4, 2013, from <http://www.forbes.com/sites/parmyolson/2012/09/06/why-estonia-has-started-teaching-its-first-graders-to-code/>.

[5] MacLaurin, M. B. (2011) The design of Kodu: a tiny visual programming language for children on the Xbox 360. In *POPL'11 Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 241–246. New York: Association for Computing Machinery.

[6] National Governors Association Center for Best Practices, Council of Chief State School Officers (2010) *Common Core State Standards (Mathematics).* Available online at <http://www.corestandards.org/Math/>.

[7] Pretz, K. (2014) Computer science classes for kids becoming mandatory. *The Institute*, IEEE, November 2014. Accessed online at <http://theinstitute.ieee.org/career-and-education/preuniversity-education/computer-science-classes-for-kids-becoming-mandatory>

[8] Resnick, M., et al. (2009) Scratch: Programming for everyone. *Communications of the ACM*, 52(11):60–67.