Transferring Skills at Solving Word Problems from Computing to Algebra Through Bootstrap

Emmanuel Schanzer, Kathi Fisler, Shriram Krishnamurthi, Matthias Felleisen
Harvard Graduate School of Education, WPI Computer Science, Brown University Computer Science,
Northeastern University College of Computing and Information Science
schanzer@bootstrapworld.org

ABSTRACT

Many educators have tried to leverage computing or programming to help improve students' achievement in mathematics. However, several hopes of performance gains—particularly in algebra—have come up short. In part, these efforts fail to align the computing and mathematical concepts at the level of detail typically required to achieve transfer of learning. This paper describes Bootstrap, an early-programming curriculum that is designed to teach key algebra topics as students build their own videogames. We discuss the curriculum, explain how it aligns with algebra, and present initial data showing student performance gains on standard algebra problems after completing Bootstrap.

Categories and Subject Descriptors: K.3.2 [Computers & Education]: Computer & Information Science Education

Keywords: algebra; skills transfer; game programming

1. INTRODUCTION

Computing educators have long argued that learning to program improves mathematics and problem-solving skills [8]. Research-based evidence for this claim has, however, been slim. While there are notable exceptions [7], a majority of projects that seek to improve problem-solving ability through programming have failed to produce measured results. Indeed, researchers versed in the computing-education literature frequently warn that programming is not known to transfer problem-solving skills to mathematics.

Transfer of learning between domains typically requires both deep structural connections between the domains and explicit instruction in how to apply concepts from one discipline in the other [1, 10, 12]. Many computing projects that target transfer to mathematical problem-solving fail to handle one of these two requirements. This raises two challenges: can we identify specific problem-solving practices in computing that have direct analogs to processes in mathematics, and can we teach them in such a way that students realize performance gains in mathematics?

SIGCSE'15, March 4–7, 2015, Kansas City, MO, USA.
Copyright © 2015 ACM 978-1-4503-2966-8/15/03 ...$15.00.
http://dx.doi.org/10.1145/2676723.2677238.

This paper describes an approach to teaching program design that measurably improves student performance on algebra word problems from standardized mathematics exams (in the USA). Our approach is embodied in a curriculum, Bootstrap, that is used in both middle- and high-school math and computing classes across the USA. This paper describes Bootstrap and how it simultaneously teaches students how to design programs and solve algebra word problems. We also present preliminary data from a pre/post-test evaluation showing student gains on algebra problems after completing Bootstrap. Our data come from students at multiple schools in multiple states in the USA; the student population includes mainstream students, not only elite ones. These data suggest that Bootstrap is achieving transfer from programming to specific topics in mathematics.

2. THE CURRICULUM: BRIDGING COMPUTING AND ALGEBRA

Bootstrap teaches students to program a videogame of their own design using algebraic and geometric concepts. Each game features a user-controlled avatar called the *player*, a *target* that the player is trying to catch, a *danger* that the player is trying to avoid, and a *background* setting in which the game takes place. In the first lesson, students "design" their games by choosing values for these elements. This customization gives students ownership and investment in the game they are trying to produce. Recent student designs vary widely: catching candy, avoiding the lunch lady, meeting aliens, collecting butterflies, etc.

Students build up their games by incrementally adding features involving their players, dangers, and targets: dangers and targets wrap-around (in side-scrolling behavior), scores rise as the player collides with the target, and the game ends when the player collides with the danger. The project is carefully designed so that each game feature teaches a particular mathematical concept. Each unit in the curriculum has three integrated components: a new feature for students to add to their games, a new programming construct or concept needed to implement the feature, and an underlying mathematics concept that relates to the programming concept. Figure 1 summarizes the curricular structure, showing the game features, programming concepts, and mathematical concepts addressed in each unit.

Bootstrap's alignment with algebra is further embodied in three aspects of our curriculum: our tools for migrating from computations to code, the shape of code that comprises a game program, and the process used to determine the required computations. We discuss each in turn.

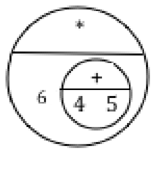
Unit	Game Feature	Programming Concept	Math Concept
1	locating elements on screen	expressions, Circles of Evaluation	coordinates
2	creating text and images	string and image operations	domain, range, kinds of data
3–5	making moving images	defining functions, examples	multiple function representations: as formulas and as tables
6	determine when game elements are off-screen	Booleans and Boolean operators	inequalities
7	responding to key-presses	conditional	piecewise function
8	collision detection	(nothing new)	Pythagorean Theorem
9	polishing games for presentation	code reviews	explaining math concepts to others

Figure 1: Curriculum structure: each unit introduces game, programming, and math concepts in parallel.

2.1 Diagramming Expressions: Circles of Evaluation

Algebra teachers often struggle to help students learn order of operations. Most teach monikers such as PEMDAS to summarize the order in which operations apply in an expression, along with phrases such as "please excuse my dear aunt sally" to help students decode the moniker. Mastering order of operations actually requires two distinct skills: understanding the structure of expressions (parsing), and applying operations in order within that structure (evaluation). Both are articulated in many state mathematics standards, but PEMDAS focuses only on the latter.

Bootstrap begins by teaching students to diagram expressions with a notation that we call "Circles of Evaluation". Assume a student is working with the expression $6 * (4 + 5)$. Bootstrap diagrams this expression as shown on the right. Each circle names a function above the line and lists arguments to the function (in left-to-right order) below the line. Nested circles correspond to nested expressions. Students learn how to evaluate a circle by reducing the innermost circle to an answer, then replacing the circle with its answer and repeating. Circles of evaluation naturally extend beyond arithmetic to other expressions that arise in programming (e.g., nested boolean expressions and computations that compose images).



From Circles to Code.

Bootstrap teaches a simple process for turning circles into syntactically-valid textual code (in a language called BSL, a pedagogically-oriented subset of Racket). We represent a circle in code by splitting it open (into a pair of parentheses) and writing its content linearly inside. The function name goes after the opening parenthesis, and the arguments follow the function name in left-to-right order. Our arithmetic circle yields the following code:

```
(* 6 (+ 4 5))
```

Circles of evaluation gently introduce students to formal notation (as many blocks-based languages do), while also drilling a challenging algebra topic. The same steps to translate circles to code apply whether the circle is for an arithmetic function, a built-in function on another datatype, or a user-defined function. This regularity, which infix syntax lacks, eases the transition from visual to textual code. Other common representations of expressions, such as abstract syntax trees, lack the visual metaphor of "opening the circle," which eases this transition.

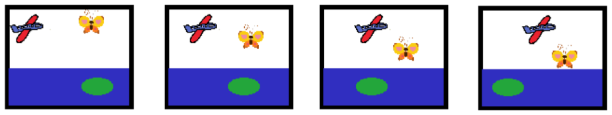


Figure 2: A videogame as a sequence of frames.

2.2 Programming Algebra: The Shape of a Game Program

Bootstrap's next connection to algebra comes from the programming model in which students develop their games. Understanding this model requires a particular perspective on how games are constructed. Figure 2 shows a game decomposed into a sequence of frames; the decomposition resembles the filmstrips that yield movies. In this game, the player uses arrow keys to move the butterfly up and down, aiming to land on the lily pad while avoiding the airplane.

In each frame of the game, each of the airplane, butterfly, and lilypad is located at a specific coordinate. The coordinates suffice to create the picture that corresponds to that frame; thus, they form the underlying *model* that characterizes individual frames (in the software engineering sense of the term, e.g., as in model-view-controller architectures). In a simple game in which each element moves in only one dimension, the model simply contains three numbers.

Between frames, the coordinates of each game element (can) change: in our example, the lilypad drifts left, the airplane flies right, and the butterfly moves according to the keys (if any) pressed by the human user. The amount by which each element's position changes between frames can be captured in a function. For example, to move the airplane rightwards across the sky, a simple linear function such as

$$f(x) = x + 5$$

computes a new x-coordinate for one frame from the x-coordinate in the previous frame.

In Bootstrap, students provide images for each of their danger, target, and player, as well as a background image (here, the blue pond); for each game element that moves, they also define a function (in code) that computes its new single-dimension coordinate from its coordinate in the previous frame. The platform iteratively produces a screenshot with the given images, updates the positions based on the functions and any key presses, then repeats, displaying the sequence of screenshots as the game. The platform also consults student-supplied functions to detect when elements have gone off screen (after which it wraps positions to the opposite edge) and to detect when game elements have collided (to raise scores or end the game).

The following examples illustrate the code that students write. The function (previously called f) that calculates the new position of the airplane is called *update-danger*:

```
(define (update-danger x)
  (+ x 5))
```

(BSL allows hyphens in function and identifier names). The *update-player* function takes an additional argument: a keypress that is used to make the player move in response to user input. The following function moves the player 3 pixels up when the up-arrow key is pressed, and 3 pixels down when down-arrow is pressed. This is an example of a *piecewise* function in mathematics. User-interactivity provides a concrete motivation for this otherwise abstract concept:

```
(define (update-player y key)
  (cond [(key=? key "up") (- y 3)]
        [(key=? key "down") (+ y 3)]
        [else y]))
```

An element has gone offscreen when its x-coordinate falls outside of the screen boundaries. Bootstrap uses this function as an opportunity to practice function composition, by having students write separate functions to check each of the left and right edges of the screen.

```
(define (onscreen? x)
  (and (onscreen-left? x) (onscreen-right? x)))
(define (onscreen-left? x)
  (> x 0))
```

Contrast to Other Early Programming Models.
These code samples illustrate how Bootstrap's underlying programming model keeps students focused on algebraic concepts. Functions are a natural and essential part of the model, not an advanced concept. The model is also interesting for what it omits relative to other early-programming platforms. Students do not use looping constructs (such as Scratch's "forever" block), mutable variables, or assignment statements. Instead, students simply write side-effect-free functions that describe how positions (and potentially other game attributes) change from one frame to the next. The lack of side effects is particularly important because it reflects the nature of functions and variables in mathematics (see Section 4). The platform, called the World framework [4], takes care of the rest.

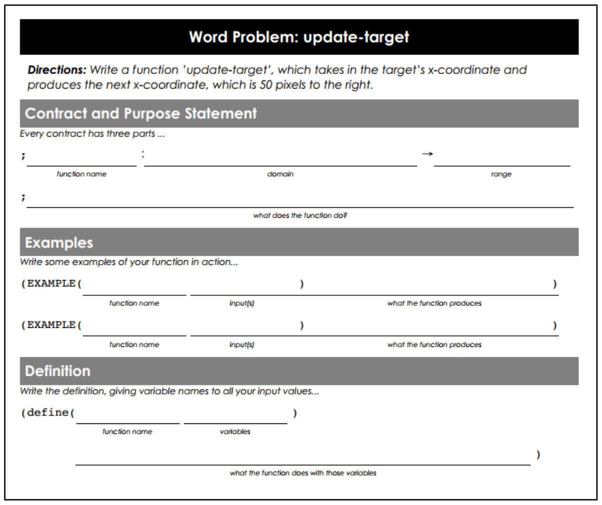
2.3 Solving Word Problems: The Design Recipe

Bootstrap's third connection to algebra arises from the process that we teach students to use when designing functions. Consider the *update-danger* function that moves the danger 5 pixels to the right. The student is taught to follow the following steps:

1. *Write a contract, specifying the domain and range for the function.* In this case, the contract is

   update-danger : number → number

2. *Write examples that illustrate the expected behavior of the function.* Examples contain actual calls to the function in BSL. Examples for *update-danger* include:



Word Problem: update-target

Directions: *Write a function 'update-target', which takes in the target's x-coordinate and produces the next x-coordinate, which is 50 pixels to the right.*

Contract and Purpose Statement

Every contract has three parts ...

; ________ : ________ → ________
function name — domain — range

; ________________
what does the function do?

Examples

Write some examples of your function in action...

(EXAMPLE(______ ______) ______)
function name — input(s) — what the function produces

(EXAMPLE(______ ______) ______)
function name — input(s) — what the function produces

Definition

Write the definition, giving variable names to all your input values...

(define(______ ______)
function name — variables

________________)
what the function does with those variables

Figure 3: A design-recipe worksheet, showing the process for solving word problems.

```
(EXAMPLE (update-danger 0) (+ 0 5))
(EXAMPLE (update-danger 5) (+ 5 5))
(EXAMPLE (update-danger 100) (+ 100 5))
```

3. *Circle changes across the examples, give a descriptive name to each independent change, then write a function whose parameters are the descriptive names and whose body is the unchanging part of the example output, substituting parameters for changes.* This case has one change: the function input and first operand to +:

```
(define (update-danger xcoord)
  (+ xcoord 5))
```

This sequence of steps, inspired by Pólya's work [11], is called the Design Recipe [3]. It helps Bootstrap students think through a function definition problem step-by-step. If they can't articulate the domain and range or write examples, they usually don't understand the problem well enough to write a correct function either. The steps thus provide teachers a valuable diagnostic when students are stuck. We use a paper-and-pencil worksheet (Figure 3) to reinforce the steps: students complete these worksheets for a variety of practice functions (including some typical mathematics word problems) as well as for every function needed for their games. Students type in and run each function that they develop through the worksheet before moving on to the next problem: this both tests their code against examples and lets them see new functionality in their games.

The design recipe is much more than a guide to writing code; it also embodies a key concept in algebra standards. Learning standards for algebra expect students to be able to work with different representations of functions. Three common representations (domain/range, input-output tables, and symbolic form) have corresponding constructs in programming (type specification or contract, test cases, and function definition, respectively). Bootstrap helps students work with all three of these representations, using the concrete context of programming to motivate when and how each representation can be helpful in problem solving. The fourth common representation, graphs, could also be supported in Bootstrap with more instructional time.

2.4 Deployment Status

Interest from teachers has been strong and growing rapidly. To date, we have offered training workshops to hundreds of teachers, most of whom are math teachers looking for ways to help students master and appreciate mathematics. Many work with students who typically struggle with math; many provide anecdotal comments about student enthusiasm and engagement in math following Bootstrap. Several thousand students have completed the curriculum, some in formal classrooms and some in after-school settings.

3. EVALUATION

Our assessment finds that Bootstrap helps students gain proficiency in certain topics in algebra. Bootstrap also helps mathematics teachers see potential connections between algebra and (some approaches to) computer science.

3.1 Teacher Perceptions

We gather feedback from teachers just after they complete one of our professional development workshops, then again after they have taught the curriculum with students at least once. Teachers' perceptions of the connections to mathematics is covered in both surveys. Many of the teachers who attend our workshops are primarily math teachers, looking to use Bootstrap to help with challenges they face in teaching Algebra I. This population does not teach Bootstrap simply for the sake of teaching computing.

In end-of-workshop surveys of 143 teachers (collected from June 2013 through August 2014), 101 indicated that they taught math. 95% of these 101 teachers strongly agreed (72%) or agreed (23%) that Bootstrap was relevant to algebra. 93% strongly agreed (74%) or agreed (19%) that Bootstrap was relevant to their own teaching. A separate survey (of in-school teachers who had used Bootstrap) confirmed that teachers see Bootstrap's relevance to math after using it: 84% (out of 32) agreed or (more often) strongly agreed with the statement "Bootstrap is an effective curriculum for teaching students math skills".

3.2 Student Performance in Algebra

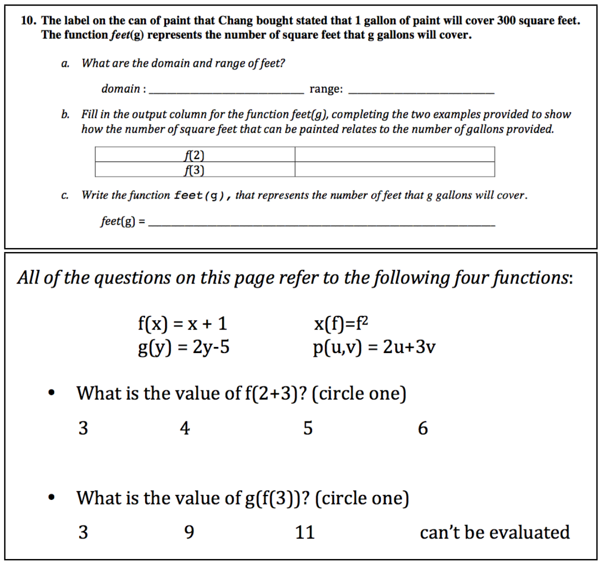
Our algebra-performance assessment focuses on (1) word problems that ask students to identify domain and range, produce input/output tables, and produce symbolic representations, and (2) problems about function composition. Concretely, we are testing the following hypotheses:

HYPOTHESIS 1. *Students who complete Bootstrap will improve in their performance on algebra word problems and function composition problems.*

HYPOTHESIS 2. *Students who complete Bootstrap will show more improvement in performance on algebra word problems and function composition problems than students who did not take Bootstrap.*

3.2.1 *The Assessment Exam*

We use a standard pre-post design to test these hypotheses. Students take a pre-test as they begin the Bootstrap module, and a post-test of similar but different questions after completing Bootstrap. Both tests are done on paper. All questions are taken from standardized algebra tests in the US state of Massachusetts (students take these tests in their 8th grade year, roughly age 13). We extend the



10. The label on the can of paint that Chang bought stated that 1 gallon of paint will cover 300 square feet. The function *feet*(g) represents the number of square feet that g gallons will cover.

a. What are the domain and range of feet?

domain : ______________ range: ______________

b. Fill in the output column for the function feet(g), completing the two examples provided to show how the number of square feet that can be painted relates to the number of gallons provided.

f(2)	
f(3)	

c. Write the function `feet(g)`, *that represents the number of feet that g gallons will cover.*

feet(g) = ______________

All of the questions on this page refer to the following four functions:

f(x) = x + 1 x(f)=f²
g(y) = 2y-5 p(u,v) = 2u+3v

- What is the value of f(2+3)? (circle one)

3 4 5 6

- What is the value of g(f(3))? (circle one)

3 9 11 can't be evaluated

Figure 4: Sample questions from Bootstrap's assessment of algebra performance. The top box shows a word problem. The bottom box shows two problems on function evaluation and composition.

state-exam questions on word problems (which ask only for symbolic representations) to ask about domain, range, and input/output examples, as those are included in state math standards (albeit emphasized less on the exams). Figure 4 shows sample questions. There are 8 word problems and 9 function-composition questions on each of the pre- and post-tests, with the latter appearing first.

3.2.2 *Student Population and Study Logistics*

The evaluation data reported here comes from computing classes at three schools: 8th grade classes at a public middle school in Massachusetts, 8th grade classes at a private school in Florida, and 9th grade classes at a private school (religiously-affiliated) in Illinois. While we have data from other schools (that show similar trends), we do not report them here: each has either too few matched pre- and post-tests, too few students for significant comparison, or insufficient IRB approval for publishing results.

Each of the teachers had completed a professional development workshop on Bootstrap. Some gathered data the first time they taught the module, while others had taught it once or twice before gathering data. Students in all classes had 30 minutes to work on each of the pre- and post tests. The same tests were used across all of the classes.

3.2.3 *Data Analysis*

We present data from a total of six classes: two from the school in Massachusetts, three from the school in Florida (one a control group that had not taken Bootstrap, labeled "Cntrl" in the tables), and one from the school in Illinois. All classes from the same state were taught by the same teacher to students at the same grade level. We present one table on each of function composition and word problems. In each table, the reported pre- and post-test score are averages across all students. In computing significance, however, pre- and post-test scores were matched for each student (rather than

Class	# Students	Pre-test	Post-test	Change
MA-fall	26	2.31	6.85	197%
MA-spring	32	2.56	5.53	116 %
FL-1	25	2.00	3.12	56%
FL-2	25	1.80	3.72	106%
FL-Cntrl	26	2.62	3.12	19%
IL	15	4.33	6.53	51%

Table 1: Pre/post-test scores on problems about function application and function composition.

Class	# Students	Pre-test	Post-test	Change
MA-fall	26	0.62	4.00	550%
MA-spring	32	1.28	3.00	134%
FL-1	25	0.80	1.72	115%
FL-2	25	0.92	1.96	133%
FL-Cntrl	26	1.54	0.81	-47%
IL	15	2.00	5.53	177%

Table 2: Pre/post-test scores on word problems.

averaged across the class), allowing us to use a two-tailed t-test to compare outcomes at the student level. Significant ($p<0.02$) gains in both function composition and word problems were found for all classes that used Bootstrap with the exception of the IL class, whose small sample size prevents us from making significance claims for the reported gains.

Table 1 reports on student performance on questions related to function application and function composition; samples of these questions appear in the lower half of Figure 4. Students earned 1 point for each problem that they answered correctly. On the per-student two-tailed t-tests, gains were significant at ranges from .003 for the Florida-1 class to 2.84e11 for the spring Massachusetts class.

Table 2 reports student performance on word problems; a sample question appears in the top half of Figure 4. Students earned 1 point for each problem on which they provided a correct function in symbolic form (part c in the sample question). On the per-student two-tailed t-tests, gains were significant at ranges from .01 for the Florida-2 class to 8.84e8 for the fall Massachusetts class.

The word-problem data reported in this paper omits student performance on the domain/range and input/output parts of the questions. State assessment exams do not include these parts, and algebra teachers often de-emphasize these representations (despite their inclusion in state standards). Reporting on only the symbolic form therefore gives us a more realistic comparison to control-group students who have not had Bootstrap. When including the results for other representations, Bootstrap students displayed far higher gains than their control group counterparts. This may speak more to the results of de-emphasizing those representations in traditional math instruction than it does to strengths of Bootstrap, but the fact that Bootstrap students were able to identify these representations on an algebra task after seeing them solely in the programming domain may strengthen the case for evidence of transfer.

The control group did not see significant gains in either function composition or word problems (word-problem scores actually dropped noticeably for this group; the teacher hypothesized that the students were less motivated for the test). We found no significant differences in pre-test scores between the control and experimental classes in Florida, and students in both conditions were taking the same mathematics class (concurrent with Bootstrap for the experimental group). This suggests that Bootstrap is having an impact beyond what students are learning in math class (perhaps affecting engagement as well as skills).

At first glance, the average scores on these tests seem rather low; on word problems, for example, some classes correctly answered fewer than 2 problems on average. For word problems, this is partly an artifact of our scoring: students answered 3 subquestions per problem (as shown in Figure 4), but we scored only one of these for this paper. Students thus answered more questions than the scores reflect. This partly explains why scores are generally higher on the function application and composition questions.

Time constraints are likely a significant factor. Word problem questions follow those for function composition, and students have only 30 minutes to complete both sections (though they may answer questions in any order). Students typically attempt no more than 5 of the 8-9 problems, yet make few errors on problems for which they write an answer. This time constraint—dictated by the participating teachers—thus appears significant in these marks being low.

The performance gains from pre-test to post-test were lower in the MA-spring courses (in both tables) than in the MA-fall courses, despite having the same teacher leading both classes. This is partly explained by higher pre-test scores in the spring class. The teacher in that course hypothesizes that the spring students had simply had more coverage of mathematics in their other classes, which could have led to higher pre-test scores.

Threats to Generalizability.

Our brief discussion of the two Massachusetts classes offers a glimpse of the many variables that differ across offerings of Bootstrap. Students could be taking Bootstrap within their math class, alongside a math class (taught by the same or a different teacher), or out of sequence with a math class. Students vary in grade levels, prior preparation in math, and confidence. Due to these (and other) variables, we cannot generalize too much from these findings. Accordingly, we do not report effect sizes and other statistical parameters.

The extra questions (on domain/range and examples) that we add to the word problems from the state exams provide a form of scaffolding that students would have to do on their own on the state exams. Additional studies must explore whether students internalize the design recipe well enough to produce correct symbolic representations in isolation.

4. RELATED WORK

While programming has had some success at developing other mathematical competencies [7], transfer to algebra has been difficult to establish. Logo was designed explicitly to teach mathematics, and many researchers have studied it as a vehicle for algebra (e.g., [5, 6]). Pea and Kurland [9] found no evidence of improved mathematical reasoning among students who had been exposed to more than 30 hours of instruction in Logo, and later questioned the entire expectation of transfer into algebra. Clements [2] found that students who mastered the concept of variables in Logo were unable to apply that understanding to algebra tasks.

Recent research suggests that deep structural similarities between thought processes in different disciplines, along with explicit instruction in those similarities, are key components to transfer [1, 10, 12]. Early-programming curricula typically lack clearly-articulated processes for problem solving (such as Bootstrap's Design Recipe), much less ones that also apply to solving problems in algebra.

In imperative programming, the terms "variable" and "function" have different meaning than in algebra. Algebra functions always return the same output for a given input (students even use the "vertical line test" to confirm this); imperative functions fail this test. Similarly, an algebra variable (called a "parameter" in programming) varies *across* function invocations, but is *constant within a given invocation.* An imperative programming variable changes its value within a function. In contrast, BSL functions and variables behave exactly like those of mathematics. Because of this relationship, students can trace function behavior in Bootstrap using *substitution* (just as they do in algebra, thereby reinforcing the idea), but this would produce the wrong answer for many Logo or SNAP functions. Wright, Rich, and Lee [14] independently deployed and assessed Bootstrap with their own instrument. Their study, which included middle- and high-school students, some experimental and some control, found significant improvements in students' understanding of variables after Bootstrap. Thomas [13] used simple BASIC programs to convey general concepts of algebraic variables, but not in the context of functions.

Bootstrap's pedagogic goals differ from those of Scratch. Scratch is designed to get students writing animation code in a matter of minutes: it focuses on rapid engagement and encouraging exploration. Bootstrap was designed to reinforce algebra: students write their first animation later than with Scratch, but having already covered key topics in mathematics standards. Its engagement is achieved differently: by having students *design* their games at the very beginning (giving them something to look forward to and repeatedly draw upon) and by using images (rather than arithmetic) to practice function composition and expression structure.

Bootstrap's Circles of Evaluation resemble the blocks that have become ubiquitous in early programming. Clearly, however, by the end of the curriculum students need to be familiar with textual notations. When to make the transition from visual to textual syntax remains open. In Bootstrap, students make the transition early (all their programming is done in text), and this has worked smoothly across several thousand middle-school students. Teachers still return to Circles periodically to emphasize expression structure (e.g., when introducing Boolean operators, or when students are confused). How these notations impact transfer to algebra remains an open research problem.

5. CONCLUSION AND ONGOING WORK

Bootstrap is a computing curriculum designed to reinforce specific learning objectives in algebra. This paper describes three key features of the curriculum and the algebraic concepts that they reinforce. Bootstrap's programming model and choice of programming language allow students to directly apply problem-solving processes for programming to standard problems in algebra. We hypothesize that these enable transfer. Preliminary data showing student performance gains in algebra after Bootstrap is encouraging. We continue to expand our data to include more teachers and more control groups.

In focusing on computing concepts that tie directly and deeply to algebra, Bootstrap also creates a gentle entry ramp for math teachers to begin teaching computing. Many countries have efforts to vastly increase the pool of qualified computing teachers, such as the CS10K project in the USA. Bootstrap appeals to content that math teachers accept as important, while simultaneously staying close enough to teachers' comfort zones that they are willing to bring it into the classroom. Several hundred math teachers completed Bootstrap professional development workshops this summer, with post-workshop surveys showing high participant confidence in their ability to begin teaching computing. This is an exciting result in and of itself as we look to expand educational opportunities in computer science.

Acknowledgements.

The NSF, Google, and Microsoft funded initial development of Bootstrap. Emma Youndtsmith and Rosanna Sobata energetically train and support teachers; Danny Yoo built the programming environment. Rachel Mark and Chris Burns helped create our earliest student workshops. Helpful reviewer comments prompted clarifications in the paper.

6. REFERENCES

[1] J. Bransford and D. Schwartz. Rethinking transfer: A simple proposal with multiple implications. In *Review of Research in Education*, volume 24, pages 61–100. American Educational Research Association, 1999.

[2] D. Clements. The future of educational computing research: The case of computer programming. *Information Technology in Childhood Education*, pages 147–179, 1999.

[3] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi. *How to Design Programs*. MIT Press, 2001.

[4] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi. A functional I/O system or, fun for freshman kids. In *ACM SIGPLAN International Conference on Functional Programming*, 2009.

[5] W. Feurzeig, S. Papert, M. Bloom, R. Grant, and C. Solomon. Programming-languages as a conceptual framework for teaching mathematics. *SIGCUE Outlook*, 4(2):13–17, Apr. 1970.

[6] R. Noss. Constructing a conceptual framework for elementary algebra through Logo programming. *Educational Studies in Mathematics*, 7:335–357, 1986.

[7] R. Noss. Children's learning of geometrical concepts through Logo. *Journal for Research in Mathematics Education*, 18(5):343–362, 1987.

[8] S. Papert. Teaching children to be mathematicians versus teaching about mathematics. *International Journal of Mathematical Education in Science and Technology*, 3(3):249–262, 1972.

[9] R. Pea and D. Kurland. On the cognitive effects of learning computer programming. *New Ideas in Psychology*, 2:137–168, 1984.

[10] D. Perkins. *Making Learning Whole*. Jossey-Bass, San Francisco, 2009.

[11] G. Pólya. *How to Solve It: A New Aspect of Mathematical Method*. Doubleday, 1957.

[12] P. J. Rich, K. R. Leatham, and G. A. Wright. Convergent cognition. *Instructional Science*, 41(2):431–453, Mar. 2013.

[13] M. O. J. Thomas. *A conceptual approach to the early learning of algebra using a computer*. PhD thesis, University of Warwick, 1988.

[14] G. Wright, P. Rich, and R. Lee. The influence of teaching programming on learning mathematics. In *Society for Information Technology & Teacher Education International Conference*, 2013.