Pinpointing Student Obstacles to Logical Reasoning about Code

Michelle Cook, Megan Fowler, Jason O. Hallstrom, Joesph E. Hollingsworth, Matthew P. Pfister, Tim Schwab, Yu-Shan Sun, and Murali Sitaraman

Technical Report RSRG-17-01
School of Computing
100 McAdams
Clemson University
Clemson, SC 29634-0974 USA

April 2017

Pinpointing Student Obstacles to Logical Reasoning about Code

Michelle Cook
College of Education
Clemson University, Clemson, SC
USA
mcook@clemson.edu

Megan Fowler
College of Education
Clemson University, Clemson, SC
USA
mefowle@clemson.edu

Jason O. Hallstrom
Engineering and Computer Science
FAU, Boca Raton, FL
USA
jhallstrom@fau.edu

Joseph E. Hollingsworth
Computer Science
IU Southeast, New Albany, IN
USA
jholly@ius.edu

Matthew P. Pfister
School of Computing
Clemson University, Clemson, SC
USA
mpfiste@clemson.edu

Tim Schwab
School of Computing
Clemson University, Clemson, SC
USA
tschwab@clemson.edu

Yu-Shan Sun
School of Computing
Clemson University, Clemson, SC
USA
yushans@clemson.edu

Murali Sitaraman
School of Computing
Clemson University, Clemson, SC
USA
murali@clemson.edu

ABSTRACT

A common method that students use to reason about the behavior of code they write is to run it using specific inputs and to study the outputs. While reasoning about code in this way is often a useful starting point, successful CS graduates eventually must be able to trace through and reason about code on all valid inputs, without relying on particular input values and without necessarily running the code. This paper describes an online system designed to pinpoint and understand the obstacles students face in reasoning about code behavior in this more sophisticated way, as well as experiments with 88 subjects who used the system in a second-year software development course. Automated analysis of results shows that the system makes it possible to identify subsets of students with varying levels and types of reasoning difficulties. Qualitative analysis highlights some gaps in the automated analysis. Initial analysis also suggests that a step-by-step reasoning process may be effective for learning.

CCS CONCEPTS

K.3.2 [Computers and Education]: Computer and Information Science Education—computer science education.

KEYWORDS

Correctness, experimentation, learning obstacles, logical reasoning, online system, tracing.

1 Introduction

A central goal of undergraduate computing education is for students to be able to reason logically and rigorously about the correctness of the software they write. Running their programs on select inputs and studying the outputs gives students only limited understanding, yet more thorough logical reasoning using paper-and-pencil methods relies on labor-intensive manual checking. Spohrer and Soloway have noted the difficulties novice students face in reasoning about compositions of statements, even as these students seem to understand individual programming constructs [28]. Over 50% of students deem the following code to be correct in an assessment exercise in a junior software engineering course in multiple offerings:

```
int max (int i, int j) {
        max = i + j;
        if (i > j) { max = max – j; }
        if (j > i) { max = max – i; }
        return max;
}
```

The error in the above code is subtle. It is not correct when i = j (even ignoring overflow/underflow errors). When the exercise is

given on a quiz or on an exam and a student deems it correct, a plausible conclusion is that the student overlooked the case when i = j. Alternatively, when a student deems the code incorrect, a plausible conclusion is that the student understood the error. Such educator conclusions are suspect unless students are asked to show detailed steps for their answers, and those steps are evaluated carefully to give feedback; see [5]. Even assuming students have a proper notion of correctness (which may be questionable [20]), what if the students who deemed the code incorrect (the expected answer) did so because they did not understand the effect of this sequence of assignment statements?

max = i + j;
max = max – j;

While the larger objective of this project is to improve the logical reasoning abilities of learners, the aim of the research experiment discussed in this paper is to help pinpoint fine-grain obstacles individual or groups face in such reasoning. Unless we understand the specific obstacles, it is hard to provide effective help. In smaller classrooms, it might be possible to accomplish this help without automation. At public institutions like ours, where some class sizes exceed 100, with serious resource constraints (such as the one used in this research with minimal TA help), automation is critical.

The focus of this paper is code composition learning obstacles that are language-independent and logical in nature. To achieve the objectives, we have built a novel online system that is supported by a verification engine[1] to help beginners reason about code. The verification engine makes it possible to learn to reason about code without assuming particular input values. It can offer a class of learner activities and directed logical feedback, not otherwise possible with standard IDEs such as Eclipse. Since correct answers do not have to be computed to check learner answers, the engine forms the basis for a powerful pedagogical agent and allows new learner activities to be created with ease.

Section 2 of the paper describes the online reasoning system and a sample reasoning activity. Section 3 contains the experimental set up. Section 4 describes quantitative results. Section 5 contains a qualitative analysis. Section 6 presents an analysis of student performance on a final exam question. Section 7 presents our conclusions and plans for further experimentation.

2 Online Reasoning System and Activities

The logical verification engine underlying the reasoning system makes it possible to create and verify activities that fall under the realm of both *tracing* questions—where students reason about given code, and *short answer coding* questions—that require students to write lines of code to meet a stated condition, in the terminology of [30]. The activities fall under the category of "correct evaluation of simple, self-referential and compound assignment operators" in [21], which includes a host of activities for each "learning objective." For example, it suggests six different objectives for assignments, some of which involve self-referential assignments. We have experimented with a set of initial activities. The reasoning system allows educators to create custom activities that address obstacles specific to their students. The activities may target various levels of Bloom's taxonomy [8] and various concepts [16][30].

The user interface of the system is divided into two sections. The left side of the screen contains an activity and reference material. The activity is a simple command that clarifies what the student is to do. The reference material serves as scaffolding for a beginner by reminding them of elements with which they might be unfamiliar. For example, it may give a list of all available relational operators, or note that ":=" is the assignment operator[2]. The right side of the interface contains the activity and a button to check the correctness of the answer and receive immediate feedback.

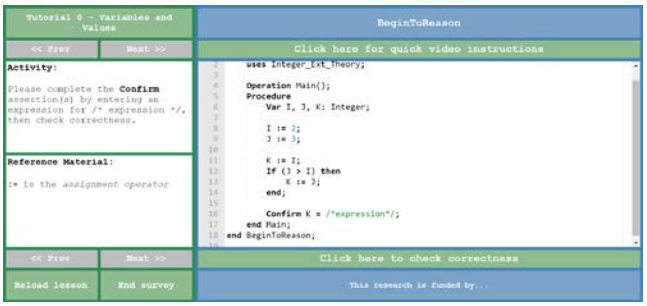

Figure 1: Reasoning System Interface Showing the Simple First Activity Using Constants

The student's task is to trace the given code and complete the *Confirm* assertion (where "=" means equals, as in logic). The student can only change */* expression */* in the assertion. This is not to be confused with an assert statement; nothing is executed. The rest of the code is locked to prevent any changes. The system is clever enough to detect and prevent trivial answers such as *"Confirm K = K"*. When the student clicks the button to check correctness, the code is sent to the verification engine, and the feedback is used to provide visual feedback for which particular answer is wrong (when there are multiple questions and answers, unlike this first example). For some activities, students are simply asked to repeat it when they get it wrong. For some others, a related, follow-up activity is given to further pinpoint which specific aspect of the activity is causing the student an obstacle.

Learners can execute code on specific inputs, and check whether their answers are correct. It is also possible to build systems that take such student inputs and outputs, runs them, and checks whether the answers match (e.g., [31]). Some systems

[1] The name of the verification engine is omitted and the references are anonymized [10][27].

[2] This interface of the reasoning system (which has multiple interface variations that can be tailored) and the rest of the paper use ":=" for the assignment operator, and = for logical equality in assertions. Also, the code uses capital letters for variable names. These interface aspects appeared to pose little or no comprehension problems to our students. The ramifications of using "=" for assignment operator, especially in late middle/early high school classes where students are also learning algebra were among the discussion points at SIGCSE 2017 following the presentation of [19].

(e.g., Problets [21]) make it possible to generate a variety of simple programs automatically and have students repeat exercises until students are performing satisfactorily. The system complements this scenario with activities such as the one below that involves reasoning over all inputs and that is only made possible with a reasoning system aided by a verification engine such as the one used in our experiment.

Since compositions are of interest, we use a trivial composition to illustrate the ideas. This activity is useful to pinpoint learning difficulties in assignment statement compositions involving self-referential assignments, and to understand subsequent tutoring mechanisms. The activity asks students to state the values of I and J in terms of the input values. These initial, input values of I and J are remembered to be #I and #J, respectively, at the line marked **Remember**[3].

```
6   Var I, J: Integer;
7   Read(I);
8   Read(J);
9
10  Remember;
11
12  I := I + J;
13  J := I - J;
14  I := I - J;
15
16  Confirm I = /* expression */;
17  Confirm J = /* expression */;
```

Figure 2: Reasoning Activity 3

Instead of stating concrete values of I and J at the end of the code for specific inputs, students now have to state the values symbolically, in terms of the initial values, namely #I and #J. We want our students to come to the general conclusion that the code swaps the values of I and J (ignoring overflows for now), and be able to explain. The (expected) correct answers are given below.

Confirm I = #J ; -- Learners can only edit the RHS
Confirm J = #I ; -- Learners can only edit the RHS

The reasoning system will attempt to confirm the answers using the verification engine. In case of legitimate claims such as the ones below, it will agree that the stated logically equivalent answers are indeed correct[4].

Confirm I = #I + #J - #I ;
Confirm J = #I + #J - #I ;

[3] This information appears in Reference Material section on the left side of the interface screen and it is not shown here. Also note that it is possible to use extra variables to hold these initial values and not use the Remember construct at all. But then learners would have twice as many names to deal with.

[4] It is important to note that the system does not need to store a set of possible correct answers because it uses a verification engine/prover. It is also important to note that that while there is no arithmetic solver in principle that would establish the validity of arbitrary such expressions, for the kinds of activities used for teaching reasoning, the verifier is more than adequate. It has been used for many years in education.

As noted earlier, the system is smart enough to reject trivial answers, such as the ones below, though they are logically correct.

Confirm I = I ;
Confirm J = J ;

Syntactically meaningless answers such as the ones below are noted to be such and students are given an opportunity to re-enter the answers.

Confirm I = I +- ;
Confirm J = $J ;

This activity is one we will discuss in much more detail in our analysis. To pinpoint the obstacles for those students who are not able to complete the activity correctly, follow-on activities are given. For example, follow-on Activity 3a has shorter code and eliminates the last assignment statement, leaving only two statements to be traced. If the student also incorrectly answers this simplified version, Activity 3b is given. It is the same as Activity 3a, except that it asks the student to assert the values after each statement. This mirrors a typical step-by-step approach to identify where exactly the student has an obstacle. A screenshot that captures Activity 3b, along with a student response is shown in the figure below. Notice that the reasoning system has highlighted (in red) which answer is wrong.

```
6   Var I, J: Integer;
7   Read(I);
8   Read(J);
9
10  Remember;
11
12  I := I + J;
13  Confirm I = #I;
14
15  J := I - J;
16  Confirm J = #I;
```

Figure 3: Activity 3b with Student Response and Visual Feedback

The example illustrates several points. Once an obstacle for a specific student or for a subset of a student population is found, the instructor may guide students directly, or the system can guide them. This guidance is not a part of the experiment in this paper.

The reasoning system has been designed to provide immediate feedback [4]. The benefit is that it prevents learners from floundering and can lead to the same level of learning in less time. If corrected immediately, students may not encode incorrect knowledge into memory [3]. The benefits of rapid (in-class) feedback are also documented in [24].

Working memory load, also known as cognitive load [29], is implicated as a major impediment to constructivist learning [9]. So the reasoning system and experiment design have been guided by related prior research [33], which discusses methods to decrease intrinsic cognitive load (e.g., sequence of simple to

complex tasks), decrease extrinsic cognitive load (e.g., segmentation, contiguity, and signaling), and promote germane cognitive load (e.g., expectancy-driven models and sub-goaling).

While we have supplemented the automated analysis with separately held student thinkalouds (as detailed in Section 5), we are incorporating screen captures and thinkalouds into the existing system. The methodology discussed in [18] will be directly useful for analyzing the screen captures.

3 Experimental Setup

The reasoning system and the activities were administered in closed lab sessions associated with a second year course on software development that emphasizes software engineering principles and uses Java. The course description includes specification and reasoning among the topics and is required for all CS majors at our institution. No formal instruction was given on reasoning before the experiment, so the tool was their first introduction to the topic. Students were allowed to ask the TAs for help if they had any difficulties in understanding what needed to be done, but few students asked for any help. The students were not time constrained. They took between 20 minutes to an hour to complete all the activities.

The experiment was conducted in fall 2016. It had benefitted from our piloting of the ideas the year before. There were five closed lab sections for the course, each with 20-25 students. There was a minor browser issue when the system was used in the first lab section, so we decided not to use any of the data from that section in our analysis. Eighty-eight students were involved in the experiments through the other four lab sections. The introductory tutorial videos (that are available now) were not available when the experiment was conducted.

4 Quantitative Data Analysis

The reasoning activities for this experiment are organized as follows. The first two are simple tutorials that familiarize the students with the layout and the basic operation of the reasoning system. Of the next eight activities used in the experiment, three involve one or more assignment statements, three involve if-then statements, and two involve while loops. We discuss in detail the analysis of student responses for Activity 3. No demographic analysis of this data is reported because the sample sizes are small and in these small sample sizes we did not notice any appreciable differences.

4.1 Student Performance on Activity 3

Activity 3 (shown in Figure 2) consisted of three follow-up activities (3a, 3aa, and 3b) as shown in the figure below. Eighty-eight students attempted this activity with the reasoning system cycling on Activity 3 until the student entered a correctly formatted answer. Once a correctly formatted answer was entered, those who answered incorrectly were moved on to Activity 3a (following the edge marked with an *I*) and those that answered correctly were moved on to Activity 4 (following edge marked with a *C*).

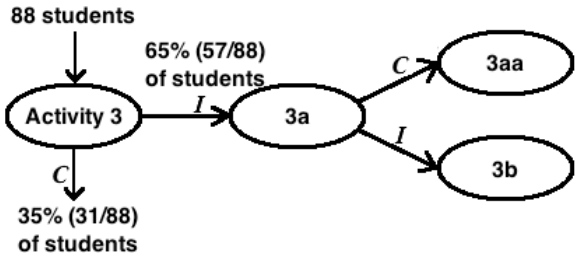

Figure 4: Students Answering 3 Correctly and Incorrectly

Eighty-eight students attempted Activity 3 with 31 correctly entering the assertions I = #J and J = #I. Sixty-five percent (or 57 out of 88 students) provided incorrect answers for one or both of the assertions, and the system moved these students on to Activity 3a. Qualitative analysis in the next section suggests possible reasons.

Activity 3a's code is shorter and it is identical to Activity 3's code except that the third assignment statement from Activity 3 has now been eliminated. Correct answers for Activity 3a are: I = #I + #J and J = #I. That is, J has the swapped value, but I has not.

By making the problem simpler and focusing on the first two assignment statements, we begin the pinpointing process. We can keep track of how many students correctly answer this sub-problem and we can begin to gauge at what level of complexity different students begin to encounter insurmountable learning obstacles. Figure 5 indicates that 49% of the 57 students who attempted Activity 3a (with two assignment statements) answered it correctly—the system then moved these students on to Activity 3aa, and those who answered 3a incorrectly to Activity 3b.

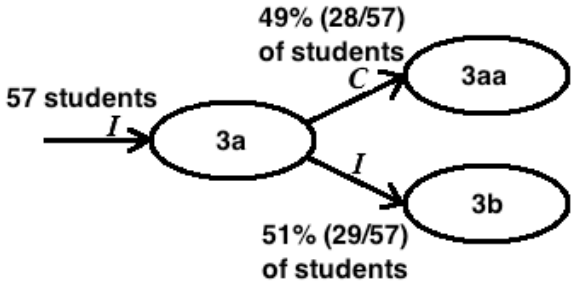

Figure 5: Further Pinpointing: Answering 3a Correctly and Incorrectly

One additional strategy incorporated into the system for pinpointing learning obstacles is to introduce step-by-step reasoning by using a *Confirm* line to cue for values of variables at intermediate points in the program. Activity 3b, illustrated through a screen shot in Figure 3, comprises the same first two assignment statements as Activity 3 and Activity 3a, but now a *Confirm* is used to cue the student for the value of the affected variable after each assignment statement. Those students who failed to correctly answer Activity 3a are being guided in Activity 3b to chunk the problem at an even more fine-grained level. The

correct answer for Activity 3b's first *Confirm* is I = #I + #J and for the second *Confirm* is J = #I.

Figure 6 shows the results of the 29 students who attempted Activity 3b. The reasoning system cycled on Activity 3b until the student was able to provide correct answers. Fifty-two percent (15/29) took one attempt to determine the correct assertions, while 17% (5/29) took two attempts to determine the correct assertions, etc. Forty-eight percent (14/29) of students took two or more tries at 3b, or approximately 15% of the original 88 students. At this point, these 15% have seen a version of this problem at a minimum of four times (in Activity 3, 3a, and at least two times in 3b), and this observation leads us to believe the following: That there is a significant percentage of students who fail to understand at a fundamental level the computation presented in these activities, and that for this particular population, the computation does not even need to involve anything more than the manipulation of integer variables through addition/subtraction and the use of assignment—i.e., in order to tease out this population, the introduction of more complex computing types, e.g., objects, is not necessary.

The qualitative analysis, discussed in the next subsection, informed us of some other learning obstacles and more specific approaches that students took to answer these activities. One of them was a surprise—a serious misconception—and would unlikely have been revealed through automated analysis alone. The other, concerning the problem solving approach that some struggling students took, was perhaps less surprising.

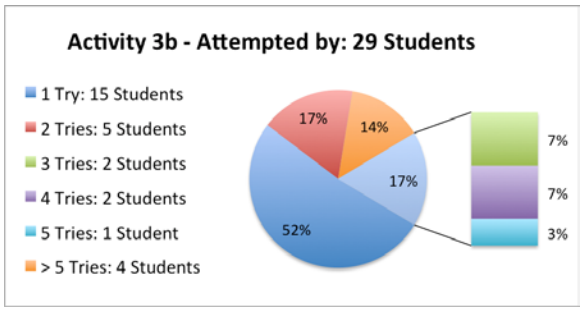

Figure 6: Most Struggling Students: Activity 3b Breakdown

Recall that students who *correctly* answered Activity 3a were moved on to Activity 3aa (see Figure 7, shown with correct student responses). Activity 3aa is identical to Activity 3a, except that the third assignment statement has been reintroduced, thus making 3aa's executable code identical to the original Activity 3 (in Figure 2). The only difference between Activities 3 and 3aa is the inclusion of step-by-step *Confirm* assertions. The driving idea behind 3aa is to observe how many of the students who demonstrated understanding of 3a (a simplified version of Activity 3) could now demonstrate an understanding of the entire three-assignment sequence found in the original Activity 3. The correct expressions for the 3aa's first two *Confirms* are exactly the same as for Activity 3a, and for the last *Confirm* it is I = #J.

```
6     Var I, J: Integer;
7     Read(I);
8     Read(J);
9
10    Remember;
11
12    I := I + J;
13    J := I - J;
14
15    Confirm I = #I + #J;
16    Confirm J = #I;
17
18    I := I - J;
19
20    Confirm I = #J;
```

Figure 7: Activity 3aa code

The analysis of the data for 3aa (Figure 8) shows that 50% (14/28) of the students were able to answer correctly in one attempt. However, 50% of the students took two or more tries at 3aa, indicating to us that there is still some amount of difficulty in understanding the fundamental constructs of addition/subtraction, assignment, and the flow of control through a sequential program. The two groups of students—the 14 students from 3aa who took two or more tries, and the 14 students from 3b who took two or more tries—added together account for 31% (28/88) of the original 88 students who attempted the sequence of activities associated with Activity 3. Many CS1 and CS2 instructors have observed overall course failure rates at this level.

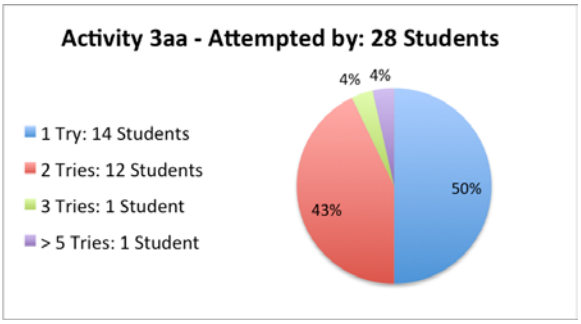

Figure 8: Students Facing Some Obstacles: Activity 3aa Breakdown

4.2 Analysis of Activity 6 with If-Then Statement

Unlike Activity 3, Activity 6 (Figure 9) which finds the minimum of three numbers, has no follow-up activities. The system simply continues to cycle on this activity until the student successfully determines the correct answer: K <= #I and K <= #J and K <= #K.

```
6       Var I, J, K: Integer;
7       Read(I);
8       Read(J);
9       Read(K);
10
11      Remember;
12
13      If I <= K then
14          K := I;
15      end;
16      If J <= K then
17          K := J;
18      end;
19
20      Confirm K /*conditional*/ #I;
21      Confirm K /*conditional*/ #J;
22      Confirm K /*conditional*/ #K;
```

Figure 9: Activity 6

Activity 6 contains if-then branching and assignment but no self-referential assignments. Figure 10 shows the difficulty this presented to the students. A full 59% of students required two or more tries at this problem with 18% requiring more than five tries.

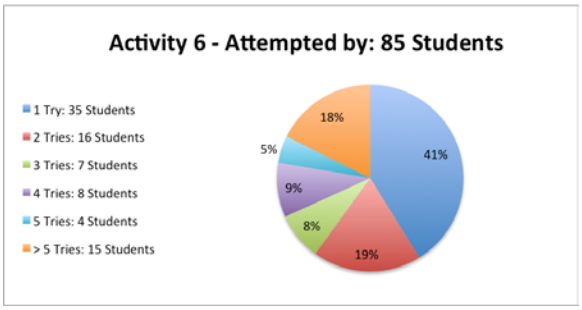

Figure 10: Activity 6 "Tries" Breakdown

Based on our experience with Activity 3, where we developed follow-up activities that cued the student to reason at intermediate steps, we now see from the analysis of Activity 6 that follow-up activities might aid in teasing out at what level the learning obstacles crop up when students reason about if-then (or if-then-else) branching.

4.3 Summary Analysis of All Activities 1-8

Figure 11 illustrates the flow through the first four activities and also includes a high-level analysis of the data. Eighty-eight students attempted activities 1–4 and the annotation "Answered correctly" indicates how many students were able to successfully complete the activity. After a successful completion the reasoning system moved the student on to the next higher numbered activity. If the student did not successfully satisfy the activity (indicated by the annotation "incorrectly"), then the student was moved on to follow-up activities. For example, with Activity 3, 35% of students answered correctly and were moved on to Activity 4, and 65% answered incorrectly and were moved on to Activity 3a.

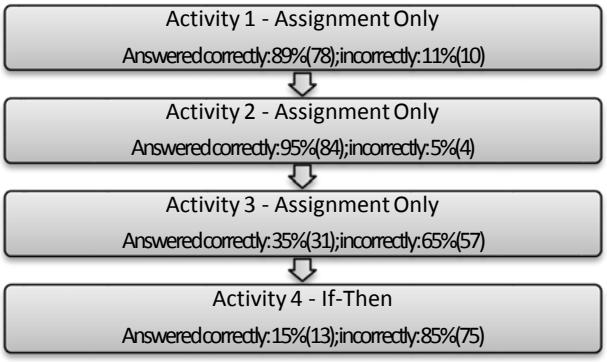

Figure 11: Summary Student Performance on Activities 1–4

Figure 12 shows the flow through activities 5–8 with 85 students attempting these activities. Unlike the first four activities, activities 5–8 had no follow-up activity for teasing out more clearly where the learning obstacles might lie. Once we understand the best approaches using the current approaches, these activities will be further refined.

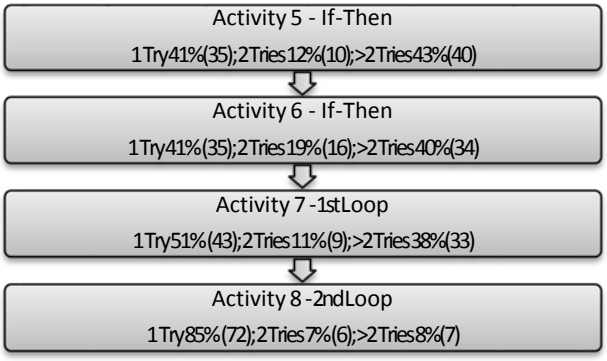

Figure 12: Summary Student Attempts on Activities 5–8

5 Qualitative Analysis

While analyzing student responses within the online reasoning system can identify potential obstacles, it is difficult to ascertain why students experience those learning obstacles without qualitative analysis. Think aloud protocols elicit what participants might be thinking, doing, and feeling, giving researchers insight into the cognitive process being used during the activities [14]. Our analysis has benefitted from the excellent discourse on using thinkalouds as a method to understand how students approach problems in [15].

After students completed the lessons in their respective lab sections using the online reasoning guide and had gone through a reasoning lecture in the class, a subset of students (n=8) volunteered to participate in a retrospective think aloud protocol several weeks later. About twenty students were invited to participate (chosen to be diverse in terms of demographics and grades), of whom eight agreed to participate. Of the eight participants, four were women, two were African American, and one was Hispanic. One student had a learning disability.

During the retrospective think aloud, participants were cued to remember the activity by completing a portion of the activities (Activities 1-3) again. In this context, it allowed the students to communicate to us if the system helped them identify when they made an error, and if they knew why they made the error, and if the system helped them move forward after the error. The retrospective think aloud also gave insight into what problem-solving strategies students were using, and what difficulties they might be encountering. Finally, as this was the first semester of data collection, the students also provided information regarding the usability of the online reasoning system and improvements to study procedures.

The think aloud happened several weeks after the students had used the system, and most students did not seem to recall the activities or answers. The findings indicate the online reasoning system helped most students pinpoint their learning obstacles. The system immediately identified when the students made an error and broke down the activity to a sub-activity or introduced steps. While most think-aloud students immediately responded with a correct answer in Activities 1 and/or 2, none of the students were initially successful on Activity 3 (shown in Figure 2).

Several students initially believed they understood the reasoning that was necessary and could quickly guess the answer, i.e., followed a guess-and-check problem solving strategy. One student shared, "I was stuck at first and [I was] one of the ones trying to guess an answer and trying to figure it out." Once the system offered a different activity after an incorrect response, the student realized it and communicated to us that the system "made me slow down" and "trace the code." Another student stated that she would rather "get a hint" and "take a stab at the same problem," rather than get the step-by-step presentation of the task. Another student agreed and said "a hint would help" or "a list of things I already put into the system."

A subset of students who made errors on Activity 3 did seem to understand the reasoning process, but made algebraic errors only because they insisted on doing it "in their heads." While we offered a sheet of paper to help with the process, only two of the eight students took up the offer. For this group, a step-by-step process would have been exactly what would be helpful.

Another subset of students had a more fundamental misconception. Their obstacle to reasoning was not revealed in the automated analysis. These students decided that the second statement that assigned a value to J was not relevant in their analysis of the value of I at the end of the code (though the third statement uses J). In other words, to reason about the value of I, they crossed out the second statement and transformed the 3-statement code in Figure 2 to the following:

I := I + J;

I := I – J;

For most of the students, the simplification of the task was advantageous in working through their obstacles. Their comments included that it was "less to think about" or "easier to process." Almost half of the students had this particular misunderstanding and were eventually able to work through it.

One student relied exclusively on guessing and checking and could not progress beyond Activity 2 within the timeframe of the meeting. While the correct answer to the question was #I + #J, the student attempted #I, #J, and #I - #J, before trying the correct answer. While guessing and checking allowed this particular student to get to the correct answer, the student never could explain why the last answer was correct. The findings indicate that presenting a problem with simplified activities and steps might be more beneficial than allowing the student to continue to guess until they get the correct answer, no matter how much they prefer that strategy.

The individual performance of the students on the reasoning activities was consistent with their overall course grade (though reasoning was only one of the many topics). The student who could not reach Activity 3 was a D/F student. The students who crossed out the second statement in their reasoning were lower B/C students. The students who erred for other reasons were A/B students. One exception was the student with the learning disability, who did as well as anyone else, though earning only a C grade in the course.

Overall, the students either stated that the activities were interesting and made them think, or were beneficial for their learning. More specifically, the simplification of the assignment statement allowed students to tease out the difficulties they were having, and in most cases, respond correctly. The simplification also discouraged an over-reliance on one type of problem-solving strategy.

6 Analysis of Learning and Methods

One fundamental question for our overall research project is if students can learn to generalize their reasoning successfully about all inputs, without concrete values. The reasoning system is an aid to assist in this learning process. Though the actual instruction on the reasoning topic involving assignment of integers was minimal, an analysis of a reasoning question on the final exam shows that most students learned to reason successfully.

Since over a 100 students were packed closely in a classroom for the final exam, students received four different variants of the exam. For this experimentation, half of the exams explicitly asked students to show the logic between each step, while on the other half, space was intentionally left between statements to allow students to write down intermediate steps, if they so wished (but they were not prompted).

All questions involved three assignment statements. The apparently harder question variation was similar to Activity 3 (but naturally involved a different sequence of three assignment statements), and the easier one used an increment, decrement, and a summation. Overall, 46 out of 58 students (nearly 75%) earned full credit on the harder question variation, and 51 out of 55 students (over 90%) earned full credit on the variants of the second and perhaps simpler question.

Since we have focused on an activity involving three assignments in this paper and also present an exam question that is similar (though the code is different), it is reasonable to ask if student performance is impacted by the choice of the question.

We note that this is unlikely to be the case because the students had seen a similar activity once several weeks before the exam, along with several other activities, and were unlikely to remember the specifics of any one activity. Furthermore, several answers included proper reasoning steps demonstrating their understanding.

The experiment of prompting for steps did not make a difference for the simpler variants, because most students likely arrived at the correct answers directly "in their heads". The rest of the section summarizes our analysis of the harder question and their variations, including potentially beneficial effects of motivation and prompting students to show their steps explicitly for code tracing and reasoning.

The bar graph below shows an analysis of how students performed when they were asked to show steps explicitly, versus when they were not asked to show work. The question was worth a total of five points, and students could earn anywhere from zero to five points.

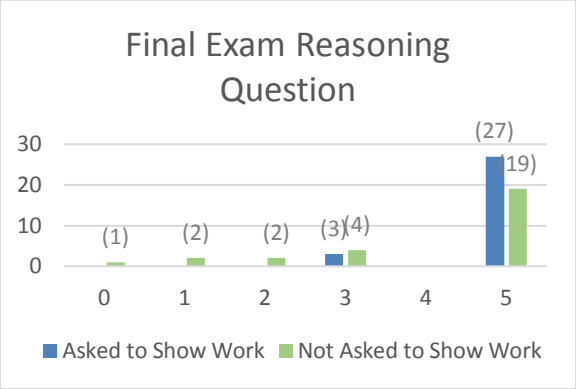

Figure 11: Comparison of Student Performance on a Reasoning Question on the Final Exam

Of the students who were asked to show work, 90% earned all five points. The remaining 10% earned three points. When students were not specifically asked to show their work, only 68% of students received full credit, with 14% earning half credit (one of two correct answers), and the remaining 18% receiving minimal or no credit.

The discrepancy in performance between these two groups did not result with lack of trying. While all 30 students in the group asked to do steps indeed did do the steps, 26 out of 28 students who were not prompted to do steps also did. So almost all students who were not asked to show steps, did some anyway. This behavior stands in contrast with the behavior noted during thinkalouds (previous section) when most students were not interested in doing steps, even when they were offered a sheet of paper and suggested they do so. Motivation (the importance of which is well understood in human psychology research) to receive a better grade on the final exam was perhaps a key factor.

How do we then explain students not doing as well when they are not required to show their work, even though many of them attempted to do so? One explanation is because of the prompt used. Students who were asked to show their work were asked to show the value of variables after each line of code. Since the other students did not receive this prompt, they used some process, but perhaps not the systematic, step-by-step process used by their counterparts. Missing this systematic process perhaps led the students who had difficulty with reasoning to underperform.

This analysis corresponds to our observations in qualitative analysis that a step-by-step process is a useful learning tool for code tracing and reasoning, especially for those who need help.

7 Discussion

We have described a system for pinpointing reasoning obstacles students face in tracing through code, and generalizing their reasoning to involve all inputs. Quantitative and qualitative analysis confirm that the system is effective. The analysis also has shown some beneficial effects on student learning through the system's feedback, though it has to be enhanced to become a tutoring system that is tailored to overcome specific obstacles.

The CS literature contains excellent online tutoring efforts (e.g.,[1][6][7][21][23]). JavaTutor is an especially ambitious effort aimed at guiding the direction of tutorials using machine learning [32]. The idea of analyzing and developing hints from past solutions of students is a key tutoring idea discussed in [26]. When advances in tutoring can be combined with the benefits offered by the verification engine, more powerful versions of tutoring that can help students generalize their code understanding and the ability to reason about all inputs can result.

We are already exploring and experimenting with several interesting questions at our institutions. We have focused on beginner code and reasoning about all inputs in the experiment, but the experiment can easily involve constants (as in Figure 1) or objects and calls to other operations. So one natural question is if activities with constants would make it possible for students to generalize their reasoning and overcome some obstacles (e.g., see [12]). At the other end, experimentation can involve tracing recursion (consider the findings in [22], for example) and loops and objects and operations to pinpoint obstacles. While this paper has limited its focus to reasoning with integers, the verification engine underlying the system is fully capable of significantly complex reasoning activities [10][11][13].

While this experiment has no involved student coding, the reasoning system will work equally well for developing activities and pinpointing obstacles when students are asked to write code for a purpose. Exploring the impact of the user interface (including use of a natural language-like syntax) and the impact of classroom instruction in conjunction with the system are other possible directions.

We are also partnering with other institutions to duplicate the experiments in different settings to confirm our findings.

ACKNOWLEDGMENTS

We acknowledge the contributions of several members of our research groups at Clemson, Florida Atlantic University, and Ohio State University. Special thanks are due to Bruce Weide. This

work is funded in part by US National Science Foundation grants CCF-1161916 and DUE-1611714.

REFERENCES

[1] V. Aleven, B. M. McLaren, and J. Sewall, "Scaling Up Programming by Demonstration for Intelligent Tutoring Systems Development: An Open-Access Web Site for Middle School Mathematics Learning," *IEEE Transactions on Learning Technologies*, 2, 2, 2009, 64-78.

[2] B. S. Alqadi and J. I. Maletic. 2017. An Empirical Study of Debugging Patterns Among Novices Programmers. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17). ACM, New York, NY, USA, 15-20. DOI: https://doi.org/10.1145/3017680.3017761

[3] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier, Cognitive Tutors: Lessons Learned, *The Journal of the Learning Sciences*, 4, 167-207, 1995.

[4] R. Azevedo and R. M. Bernard, "A Meta-Analysis of the Effects of Feedback in Computer-Based Instruction", *Journal of Educational Computing Research*, 13(2), 111 – 127, 1995.

[5] N. C. C. Brown and A. Altadmri. 2014. Investigating novice programming mistakes: educator beliefs vs. student data. In Proceedings of the tenth annual conference on International computing education research (ICER '14). ACM, New York, NY, USA, 43-50. DOI: http://dx.doi.org/10.1145/2632320.2632343

[6] P. Bhattacharya, L. Tao, B. W. Kai Qian, and E. K. Palmer, "A Cloud-based Cyberlearning Environment for Introductory Computing Programming Education," In 2011 11th International Conference on Advanced Learning Technologies (ICALT), 2011, 12-13.

[7] P. Bhattacharya, L. Tao, Y. Fu, and K. Qian, "A Collaborative Interactive Cyber-learning Platform for Anywhere Anytime Java Programming Learning," In 2011 11th International Conference on Advanced Learning Technologies (ICALT), 2011, 14-16.

[8] B. S. Bloom, M. D. Engelhart, E. J. Furst, W. H. Hill, D. R. Krathwohl (Eds.) Taxonomy of Educational Objectives. The Classification of Educational Goals, Handbook I: Cognitive Domain. New York, David McKay Company, Inc., 1956.

[9] M. P. Cook, "Visual representations in science education: The influence of prior knowledge and cognitive load theory on instructional design principles," Science Education 90, 6, 2006, 1073-1091.

[10] C. T. Cook, H. Harton, H., Smith, and M. Sitaraman, "Specification Engineering and Modular Verification Using a Web-Integrated Verifying Compiler", In 2012 34th International Conference on Software Engineering (ICSE), 2012, 1379-1382.

[11] C. T. Cook, S. Drachova-Strang, Y-S. Sun, M. Sitaraman, J. C. Carver, and J. E. Hollingsworth,, "Specification and Reasoning in SE Projects Using a Web IDE", In Proceedings Conference on Software Engineering Education & Training (CSSE&T), 2013

[12] D. De Bock, J. Deprej, W. van Duren, M. Roelens, and L. Verschaffel, "Abstract or Concrete Examples in Learning Mathematics? A Replication an Elaboration of Kaminsky, Stoutsky, and Heckler's Study," Journal for Research in Mathematics Education 42, 2, 2011, 109-126.

[13] S. Drachova-Strang, J. O. Hallstrom, J. E. Hollingsworth, J. Krone, R. Pak, and M. Sitaraman, "Teaching Mathematical Reasoning Principles for Software Correctness and Its Assessment," ACM Transactions on Computing Education 15, 3, Article 15 (August 2015), 22 pages. DOI=http://dx.doi.org/10.1145/2716316..

[14] K. A. Ericsson and H. A. Simon. How to study thinking in everyday life: Contrasting think-aloud protocols with descriptions and explanations of thinking. Mind, Culture, and Activity, 5(3), 178-186, 1998.

[15] S. Fitzgerald, B. Simon, and L. Thomas. 2005. Strategies that students use to trace code: an analysis based in grounded theory. In Proceedings of the first international workshop on Computing education research (ICER '05). ACM, New York, NY, USA, 69-80. DOI=http://dx.doi.org/10.1145/1089786.1089793

[16] K. Goldman, P. Gross, C. Heeren, G. Herman, L. Kaczmarczyk, M. C. Loui, and C. Zilles, "Identifying important and difficult concepts in introductory computing courses using a Delphi process," In Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, 2008, 256-260.

[17] P. B. Henderson, "Mathematical Reasoning in Software Engineering Education," Communications of the ACM 46, September 2003, 45-50.

[18] C. D. Hundhausen, J. L. Brown, S. Farley, and D. Skarpas. 2006. A methodology for analyzing the temporal evolution of novice programs based on semantic components. In Proceedings of the second international workshop on Computing education research (ICER '06). ACM, New York, NY, USA, 59-71. DOI=http://dx.doi.org/10.1145/1151588.1151599

[19] T. Kohn. 2017. Variable Evaluation: an Exploration of Novice Programmers' Understanding and Common Misconceptions. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17). ACM, New York, NY, USA, 345-350. DOI: https://doi.org/10.1145/3017680.3017724

[20] Y. B-D. Kolikant. 2005. Students' alternative standards for correctness. In Proceedings of the first international workshop on Computing education research (ICER '05). ACM, New York, NY, USA, 37-43. DOI=http://dx.doi.org/10.1145/1089786.1089790

[21] R. Kumar, C. P. Rose, Y. Wang, M. Joshi, and A. Robinson, "Tutorial Dialog as Adaptive Collaborative Learning Support," In Proc. 2007 Conference on Artificial Intelligence in Education: Building Technology Rich Contexts That Work, IOS Press, 2007, 383-390.

[22] C. M. Lewis. 2014. Exploring variation in students' correct traces of linear recursion. In Proceedings of the tenth annual conference on International computing education research (ICER '14). ACM, New York, NY, USA, 67-74. DOI: http://dx.doi.org/10.1145/2632320.2632355

[23] C. Li, Z. Dong, R. Untch, Chasteen, and N. Reale, "PeerSpace—An Online Collaborative Learning Environment for Computer Science Students," In 2011 11th International Conference on Advanced Learning Technologies (ICALT), 2011, 409-411.

[24] K. Martinez and M. Eisenhart, L.E.A.P. – Literature Review of Best Practices in College Physics and Best Practices for Women in College Physics, January 2004; available at http://advance.colorado.edu/research_best_practices.html.

[25] R. Moreno, "Decreasing Cognitive Load for Novice Students: Effects of Explanatory versus Corrective Feedback in Discovery-Based Multimedia," Instructional Science (32), 2004, 99-113.

[26] T. W. Price, Y. Dong, and D. Lipovac. 2017. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17). ACM, New York, NY, USA, 483-488. DOI: https://doi.org/10.1145/3017680.3017762

[27] M. Sitaraman, B. Adcock, J. Avigad, D. Bronish, P. Bucci, D. Frazier, H. M. Friedman, H. Harton, W., Heym, J., Kirschenbaum, J., Krone, H. Smith, and B. W. Weide., "Building a Push button RESOLVE Verifier: Progress and Challenges," In Formal Aspects of Computing, Springer, 2011, 607-626..

[28] J. C. Spohrer, and E. Soloway, "Novice mistakes: are the folk wisdoms correct?" Communications of the ACM 29, 7, July 1986, 624-632.

[29] J. Sweller, P. Ayres, and S. Kalyuga, Cognitive load theory, 2011, New York, NY, Springer.

[30] A. E. Tew, Assessing fundamental introductory computing concept knowledge in a language independent manner, Ph.D. Dissertation, Georgia Institute of Technology, 146 pages.

[31] Web-Cat, 2017. Web-CAT is an advanced automated grading system that can grade students on how well they test their own code. http://web-cat.org/home

[32] J. B. Wiggins, K. E. Boyer, A. Baikadi, A. Ezen-Can, J. F. Grafsgaard., E. Y. Ha, J. C. Lester, C. M. Mitchell, and E. N. Wiebe, "JavaTutor: An Intelligent Tutoring System that Adapts to Cognitive and Affective States during Computer Programming," In Proceedings of the 46h SIGCSE Technical Symposium on Computer Science Education, ACM Press, 2015, 599-599.

[33] P. Wouters, F. Paas, and J. J. G. van Merriënboer, How to Optimize Learning from Animated Models: A Review of Guidelines Based on Cognitive Load, Review of Educational Research, 2008; DOI: 10.3102/0034654308320320