Network Visualization and Assessment of Student Reasoning About Conditionals

Nathan Hurtig[*]
Joseph Hollingsworth[*]
hurtigna@rose-hulman.edu
hollings@rose-hulman.edu
Rose-Hulman Institute of Technology
Terre Haute, IN, USA

Sarah Blankenship[*]
Eileen Kraemer[*]
Murali Sitaraman[*]
smblank@clemson.edu
etkraem@clemson.edu
msitara@clemson.edu
Clemson University
Clemson, SC, USA

Jason O. Hallstrom
jhallstrom@fau.edu
Florida Atlantic University
Boca Raton, FL, USA

ABSTRACT

Understanding the thought processes of students as they progress from initial (incorrect) answers toward correct answers is a challenge for instructors, both in this pandemic and beyond. This paper presents a general network visualization learning analytics system that helps instructors to view a sequence of answers input by students in a way that makes student learning progressions apparent. The system allows instructors to study individual and group learning at various levels of granularity.

The paper illustrates how the visualization system is employed to analyze student responses collected through an intervention. The intervention is BeginToReason, an online tool that helps students learn and use symbolic reasoning—reasoning about code behavior through abstract values instead of concrete inputs. The specific focus is analysis of tool-collected student responses as they perform reasoning activities on code involving conditional statements.

Student learning is analyzed using the visualization system and a post-test. Visual analytics highlights include instances where students producing one set of incorrect answers initially perform better than a different set and instances where student thought processes do not cluster well. Post-test data analysis provides a measure of student ability to apply what they have learned and their holistic understanding.

CCS CONCEPTS

• **Human-centered computing → Graph drawings**; **Information visualization**; **Visualization systems and tools**; **Empirical studies in visualization**.

KEYWORDS

Graph Drawing, Information Visualization, Visualization Systems and Tools, Empirical Studies in Visualization, Program Verification

[*]Authors contributed equally to this research.

ACM Reference Format:
Nathan Hurtig, Joseph Hollingsworth, Sarah Blankenship, Eileen Kraemer, Murali Sitaraman, and Jason O. Hallstrom. 2022. Network Visualization and Assessment of Student Reasoning About Conditionals. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol 1 (ITiCSE 2022), July 8–13, 2022, Dublin, Ireland*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3502718.3524793

1 INTRODUCTION

Rising enrollment, especially in computer science, has motivated a need to develop, explore, and use new educational methods and online tools in classrooms [8]. Specifically, automated code checking systems have seen increased use [13, 18, 19]. Also, the use of peer assessment and instruction has become more common to increase engagement in learning and minimize graduate assistant and instructor usage [4, 27, 32].

The trend toward hybrid and remote learning in computer science at many institutions [7] has accelerated with the arrival of the pandemic. Educators see the potential benefits of online teaching and the challenges in student motivation and student-faculty interaction [1, 3]. There is now more reason to implement and use online tools.

The data analysis in this paper focuses on a challenging topic for most students to learn: Reasoning effectively about the behavior of code involving conditional statements. The novelty of the symbolic reasoning approach is that it goes beyond understanding code behavior on specific test inputs and helps students generalize their reasoning using abstract values and achieve a holistic understanding [16]. The intervention to aid students in learning is an online reasoning tool, BeginToReason, that has been shown to be effective in helping students to reason about sequential code[11, 17]. The tool collects student responses as they learn reasoning and progressively develop more correct answers.

To gain the full benefits of interventions such as the reasoning tool, ultimately, instructors need a way to understand the learning progressions of their students. To aid in this objective, we have developed a novel network visualization learning analytics system. Figure 1 shows the relationships. The right half of the figure with blue arrows shows the usage of the BeginToReason tool. The left half adds the analytics system with green arrows. The system helps instructors understand and analyze the difficulties their students face, individually or collectively. The system is applicable to student learning of any topic in any environment where intermediate and possibly wrong answers are collected for analysis as students progress toward correct answers.

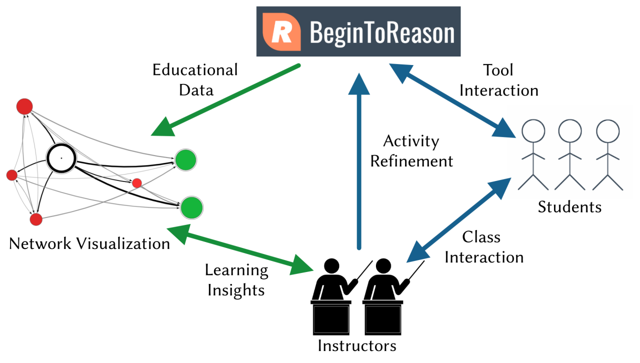


Figure 1: Visualization system and reasoning tool.

1.1 The Intervention

To be effective programmers, computer science students need to learn to reason about the behavior of the code they write. One common way students learn to do such reasoning is to run their code or hand execute their code on a variety of fixed test inputs and generalize their reasoning from these specific cases. While such reasoning is an appropriate starting point, to predict the behavior of code with confidence and gain a more thorough understanding, students need to be able to generalize their analysis. For example, they should be able to reason that a given program computes a function of some inputs `x` and `y` or that it computes the minimum of two given inputs. Prior research has established that a vast majority of undergraduate students can learn to do such symbolic reasoning through a combination of instruction and tool usage.

The objective of the intervention described in this paper is to help students reason symbolically about code that involves conditional statements. The specific intervention is the use of an online tool, BeginToReason. The tool incorporates a variety of reasoning activities, beginning with straight-line code to code involving `if` and `while` statements. Students receive a single lecture on the basics of symbolic reasoning before using the tool in a lab setting. The lecture focuses on reasoning about assignment statements symbolically, leaving students to learn to reason about conditional statements from the tool and its activities. Overall, the intervention is set up so that it is easy to adapt for any institution interested in teaching the ideas.

A key element of the reasoning tool that is particularly important for this paper is that student responses are free-form, and all responses are recorded. A student may attempt to solve a problem many times, get tool feedback, and continually improve their answers.

1.2 Visualization and Assessment of Student Learning

The network visualization learning analytics system we have developed is motivated by prior research, such as the work in [5]. Our system automates the presentation of collected data of intermediate and final answers, both right and wrong. In the underlying learning network that is visualized, each node in the directed graph that forms the network represents an answer input by some students, and an edge to another node shows a subsequent answer.

The visualization system is designed to have an intuitive interface. It allows instructors to study how students approach problem-solving and make progress. It helps them identify common difficulties and individuals or groups of students with similar difficulties. It is interactive so that instructors can adjust the level of granularity at which the same data is visualized.

Along with using the learning analytics system, our assessment also employs a post-test. The post-test includes multiple-choice questions to gauge students' ability to apply what they have learned and free response questions to capture their level of understanding and the formality with which they can express that understanding. A key goal is a holistic understanding, whereby for example, in reasoning about a piece of code that computes a minimum of given numbers, a student can realize and articulate that it computes a minimum and not just its mechanics.

2 RELATED WORK

Unlike other tutors and IDEs, the reasoning tool used in our intervention is backed by the well-known RESOLVE verifier [10, 33]. It is one of the several different existing verification tools summarized in [28]. The verifier allows the tool to facilitate symbolic reasoning over abstract input values. In turn, the tool allows students to input free-form assertions in their responses to reasoning questions. The tool collects some user data for research into student thinking. For over a decade, thousands of undergraduate students at multiple institutions have employed symbolic reasoning approaches using this reasoning tool in CS courses [12, 23, 26] and software engineering projects [30].

Trustworthy software engineering undoubtedly requires logic and reasoning rooted in mathematics that many students are taught in a simple computer science curriculum [24]. Coding to test cases, which most students do now, may lead to a lack of understanding of the assignment [15], so a holistic reasoning approach can be more helpful. Also, the logic employed in symbolic reasoning has the potential to serve the students well in higher-level classes and their careers with algorithm design and optimizations [22]. Symbolic reasoning helps students see programming and mathematics as more integrated [21]. Reasoning tools similar to the one used in this research have shown to improve both performance and attitudes towards reasoning elsewhere [34].

Learning progression, an idea that motivates this paper, is based on the observation that learning for many skills tends to follow an expected path (progression), in which more complex skills are built on foundational skills (e.g., sitting, then crawling, then walking) [25]. A learning progression describes how the skills might be demonstrated in both early and increasingly advanced forms. Instructors who can identify demonstrations of these skills in their students can provide appropriate experiences and levels of challenge to guide students along such a progression. For example, educators in science have identified learning paths in traditional academic learning about science concepts and used these to structure curricula [35]. While learning "trajectories" for computing concepts have been identified for the K-8 space [31], we do not know of such progressions or trajectories for *abstract* reasoning about conditionals or other specific programming constructs. The network visualization we describe here is an important step in informing instructors.

A distinguishing aspect of the visualization system is that it enables educators to note their own trends in data at a high level but still draw their own conclusions through close inspection. This goal differs from approaches such as the Error Quotient [37], the WatWin measure [29], the Normalized Programming State Model [9], and machine learning approaches [2, 36] in that this research seeks to provide educators the means for network visual analytics, rather than analyze data for them.

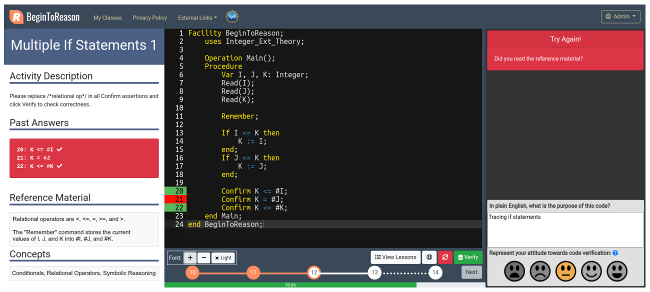
Prior research has employed networks to analyze and visualize student learning data. For example, in [14], a graphical model of how students navigate online courses is found. Prior research has also explored using networks to represent how abstract concepts are related [20] and how learned concepts progress [38]. But the following central ideas in the visualization system make it novel. It models students in a much smaller scope, and instead of analyzing events within courses or concepts, it analyzes answers within individual questions. It uses a dynamic and interactive visual presentation, unlike many summarized in [39].

3 TOOL OVERVIEW

The intervention described in this paper is the use of a reasoning tool called BeginToReason. A screenshot showing a tool activity appears in Figure 2. The tool's interface comprises three main panels. The middle panel displays an activity where the student directs most of her focus. The task at hand is to analyze the code in the operation and determine what the code does. In the code, ":=" is used to distinguish assignments from equality in mathematical assertions embedded with code. Students learning Java appear to have no difficulty with such minor syntactic variations.

A student accomplishes the task by filling in three `Confirm` assertions at the end of the code segment to capture what the code does with the given variables. Initially, the assertions appear empty. For this example, there are three `Confirm` assertions to be filled, each of which requires the student to enter a relational operator. To successfully complete the task, all three assertions at the end must verify to be true regardless of input values. In this symbolic reasoning, before the conditional statement, input values of variables `I`, `J`, and `K` are remembered to be `#I`, `#J`, and `#K`, respectively on line 11 with the `Remember` keyword. Current values of variables at any state are expressed using just their names. For example, when `K` is used at the end of the code, undecorated with `#`, it denotes `K`'s value at the end.

The left panel contains a plain English description of the current activity, a record of past answers to help students recall their previously entered answers, a reference material area that provides scaffolding, and a list of concepts related to the activity. The right panel provides feedback to the student for the correctness of the answers supplied, a student response area where instructors can request answers to free-response or multiple-choice questions, and an area to obtain attitudinal responses from the student.



Multiple If Statements 1

Activity Description

Please replace /*relational op*/ in all Confirm assertions and click Verify to check correctness.

Past Answers

```
20: K <= #I ✓
21: K = #J
22: K <= #K ✓
```

Reference Material

Relational operators are <, <=, =, >=, and >.

The "Remember" command stores the current values of I, J, and K into #I, #J, and #K.

Concepts

Conditionals, Relational Operators, Symbolic Reasoning

```
1  Facility BeginToReason;
2      uses Integer_Ext_Theory;
3
4      Operation Main();
5      Procedure
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 <= #I;
21         Confirm K = #J;
22         Confirm K <= #K;
23     end Main;
24 end BeginToReason;
```

Try Again!

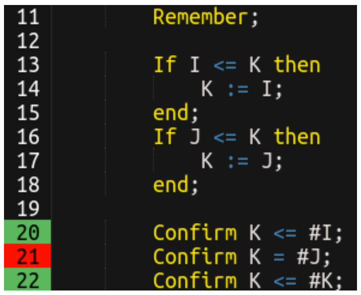
Did you read the reference material?

In plain English, what is the purpose of this code?

Tracing if statements

Represent your attitude towards code verification:

Figure 2: A screenshot of a BeginToReason activity.



```
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 <= #I;
21         Confirm K = #J;
22         Confirm K <= #K;
```

Figure 3: Middle panel shown with a student's assertions.

For this particular activity, the relational operators that cause these assertions on lines 20, 21, and 22 to evaluate to true are <=, <=, <=, respectively. In other words, the conditional statement makes variable K's value to be the minimum value of the three input values. In Figures 2 and 3, a student has entered correct responses only for lines 20 and 22 and received exactly that feedback through red and green color highlighting. (The tool allows the colors to be changed for accessibility). All student responses are saved, and they are allowed to retry indefinitely.

4 NETWORK VISUALIZATION

This section summarizes the network visualization learning analytics system that we have developed to study student responses as a network. When tackling an activity, such as a code reasoning activity shown in the previous section, students will often try many incorrect answers before finding a correct solution. The system helps visualize not just one sequence of intermediate answers proposed by one student but all sequences generated by a group of students in a class. A sequence is a list of chronologically ordered responses given by one student while attempting to answer a question. The visualization attempts to provide insight into how students think by focusing on how students progress between answers. The visualization system can be used in *any* learning scenario that fulfills these criteria:

(1) Students attempt to answer a question and repeat until they input a correct answer.
(2) The state space of reasonable answers is limited.
(3) Allowed formats of answers are of the form of fill-in-the-blank, multiple-choice, or calculations.

These criteria are general enough so that the visualization system can be applied to educational contexts outside of an online introduction to conditional statements. Most online activities with instant feedback fulfill the system's criteria regardless of their topic.

Consider a reasoning activity such as in the previous section where a student is asked to fill in a `Confirm` assertion. A fictional student "Alex" might have tried the intermediate incorrect answers `J = 1` and `J < 0`, and then found `J <= 0`, which we suppose is a correct answer. The visualization system combines Alex's sequence with the sequences of answers from other students into a network. To illustrate these ideas, we show below a small fictional input data set in Table 1 and its corresponding network visualization in Figure 4.

Table 1: Synthetic Raw Data

Student Identifier	Student's Attempt	Attempt Correct?
Alex	`J = 1`	Incorrect
Alex	`J < 0`	Incorrect
Alex	`J <= 1`	Correct
Derek	`J = 1`	Incorrect
Derek	`J <= 0`	Correct
...	...	...
Emma	`J <= 0`	Correct

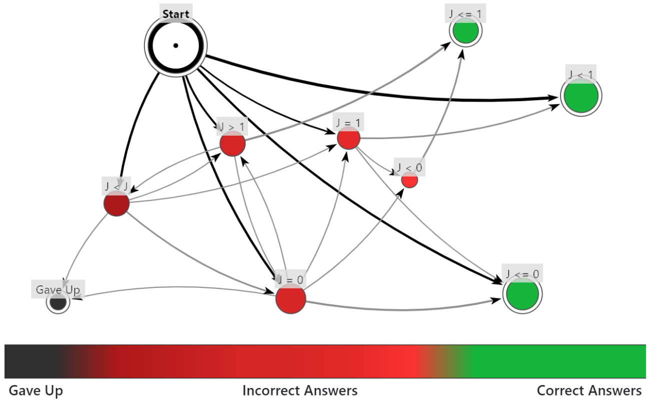


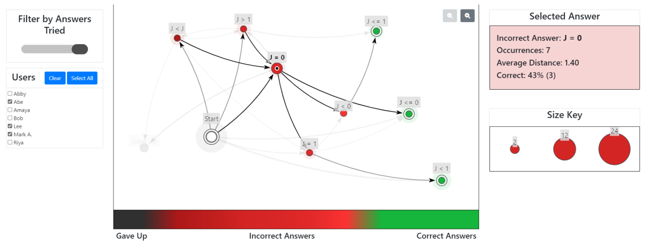
Figure 4: Network visualization of the above synthetic data.

There is a corresponding node (circle) in the network for every unique answer tried by at least one student. Incorrect answers are in red, and correct answers are in green. The color scheme can be changed from red-green to yellow-blue to enhance accessibility. The black node denotes that a student "gave up" on solving the problem and did not find a correct answer. Answers that many students attempted are larger in the network, so common answers are more noticeable. Likewise, edges grow wider as more students input the same two answers in a row. The paths of all students through the network begin at the "Start" node and end either at a correct answer or at a node where they gave up. Double circles highlight the beginning and end nodes of the network.

The visualization uses horizontal positioning to establish a hierarchy of answers. Correct answers are on the right side of the network, and the "Gave Up" node is on the left. The remaining intermediate incorrect answers are placed according to their calculated *distance* from the correct answers. Each node's distance metric is calculated as the arithmetic mean of how many more attempts students tried after leaving an incorrect answer before reaching a correct answer. Nodes with higher distances are positioned on the left side of the network, farther away from the correct answers. Their color is also slightly darkened to visually indicate that students made more attempts to arrive at a correct answer following answers represented by these darker nodes.

The network in Figure 4 is only a static screenshot of the online visualization system. The network is a dynamic physics simulation powered by `D3.js` [6]. The simulation pulls connected nodes closer together and pushes unconnected nodes apart. Users can click and drag nodes in the network to arrange them as they see fit.

The tool offers options and filters that allow educators to configure the visualization dynamically. When a user selects a node, the tool displays the node's calculated distance from the correct answers, how many students answered the question correctly immediately following their input of the selected attempt, and a list of the students that input the answer. A screenshot of the interface is in Figure 5. Users can direct the tool to filter by students to show only a subset of the classroom's paths through the network. In Figure 6, a single student's path stands out because all other paths and nodes have been dimmed. Users can also combine nodes through drag-and-drop, so they can simplify the network by aggregating similar answers together by their concepts.



Incorrect Answer: **J = 0**
Occurrences: 7
Average Distance: 1.40
Correct: 43% (3)

Figure 5: User interface of the network visualization.

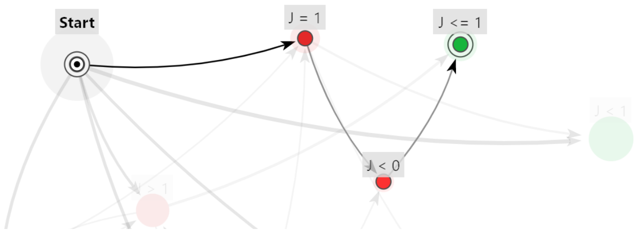


Figure 6: A single student's path through the network.

5 EXPERIMENTAL SETUP

The data used for analysis in this paper was collected from an experiment conducted in 2021 at one of our institutions. 76 out of 94 total students in a second-year software development course consented to participate in the study. The class uses Java, and students are expected to have learned foundational object-oriented programming concepts in a prerequisite course.

Students received an in-person lecture on formal reasoning for the experiment, focusing primarily on reasoning about a straight-line sequence of assignment statements. The students completed activities in a lab using the symbolic reasoning tool the week after the lecture. They completed a post-test on symbolic reasoning a few days later. 66 out of 76 of the consenting students also took the post-test. The post-test asked students to choose the correct assertions for a given code segment, similar to the activity in Section 3. The post-test also asked students to choose the correct code segment to make the given assertions true. Both kinds of assessments were presented on each style of post-test question to check if students were able to apply their understanding of conditional code reasoning beyond the style of questions they had seen in the activities.

6 ANALYSIS

6.1 Network Visualization Learning Analytics

This section presents examples of qualitative insights gained from employing the network visualization learning analytics system on the tool-collected data from symbolic reasoning lab activities.

In the same way that one specification can be realized by multiple code segments, multiple assertions may be true at the end of a given code segment. Students who reach different correct answers may do so because they employed different thinking strategies. These differences can be detected using the interactive network display.

As part of the tool's lab activity, students were asked to complete the `Confirm` assertions found in Listing 1. The code takes an integer I, and if I is non-negative, sets J to be -I. Otherwise, I is negative, and J is set to be I. Unlike a minimum-finding example, this example is designed to let students explore more. Students reached two distinct correct answers for this problem: "J <= 0" (Answer A) and "J <= I" (Answer B). Whereas the first answer, which more closely corresponds to the purpose of the code, was the intended answer to demonstrate a holistic understanding, the less intuitive second answer is also correct. Of the 99 students that attempted the lesson, 51 reached Answer A and 47 reached Answer B.

Listing 1: Lab activity code.

```
Read(I);
If I >= 0 then
    J := - I;
else
    J := I;
end;
Confirm J /*relational op*/ /*expression*/;
```

The networks shown in Figure 7 were made using the network visualization's selection feature, with Figures 7a and 7b showing the network generated by Groups A and B, respectively. They show a clear visual difference between the thought processes: Students who

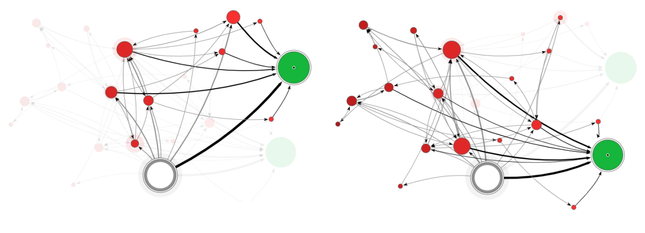

(a) Answer A (b) Answer B

Figure 7: Comparison between two correct answers.

reached different correct answers tended to not share intermediate incorrect answers along the way, and the students in Group B, who picked the less intuitive answer, visited the higher-distance nodes. This observation is supported by quantitative analysis as well: Students in Group A reached the correct Answer A after an average of 0.824 incorrect answers, but students in Group B reached the correct Answer B after an average of 1.447 incorrect answers.

Figure 8 shows a network generated from another question in the lab activity. The undirected, messy network shows that students did not have many thinking strategies in common, though they all arrived at the same correct answer.

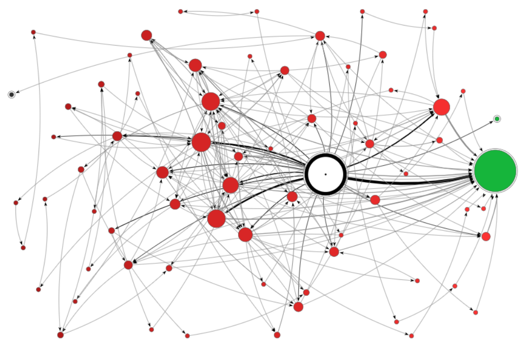


Figure 8: A cluttered network means answers were diverse.

This network shows that some students employed a "guess-and-check" strategy on this problem. Supporting this theory is the relative "distance" of the Start node: many incorrect answers are farther away from the correct answer than the Start node. Since the distance of the Start node is mathematically the average number of attempts across all students, this means that all the incorrect answers to the left of the Start node are worse off than the class average. This range might be due to a bifurcation in students: those who got the correct answer through formal thinking and those that relied on guessing. This claim is supported quantitatively: 44% of the 98 students reached the correct answer on their first try. However, the remaining 56% took an average of 5.291 attempts before reaching the correct answer.

Insights such as the ones here, made possible through network visualization, can help educators in two ways. First, they can serve as an early warning that some students may not fully understand the material. Detecting guessing methods in labs or homework allows educators to respond and adjust proactively before administering summative assessments like exams. Second, it can inform educators that the question may be too difficult or exploratory. It is challenging for educators to measure the difficulty of questions that allow students to retry after incorrect answers, as most students might eventually guess correctly. By providing a way to visualize guessing, network analysis can help educators gauge the difficulty of their questions and arrive at questions that better channel students' thought processes.

6.2 Post-Test Data Analysis

In the multiple-choice questions on the post-test, students either selected assertions that satisfied a given code segment (similar to the lab activity) or selected a code segment that would make given assertions true. Figure 9 shows the average score versus the difference between the two types of questions. The graph's x-axis is the average score on the conditional questions, and the y-axis is the difference between scores on choosing code segment questions and choosing assertions questions. Students maintained the same performance on both kinds of questions. More points are clustered towards the higher averages and zero or positive differences, showing that students are learning to reason about code in both directions. The p-value for this comparison of scores is 0.048, indicating that the data are statistically significant.

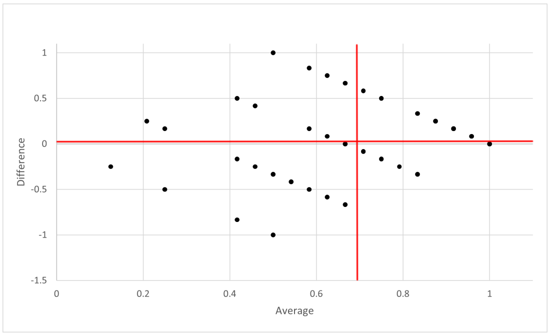


Figure 9: Matched-pair analysis for selecting assertions versus code segments.

The post-test questions, aside from differing in selecting assertions or code segments, were further subdivided into three categories: The answers for each question would either be "holistic" (e.g., code computes the maximum), "holistic/formal" (e.g., `K = Max(#I, #J)`), or "formal" (e.g., `K >= #I` and `K >= #J`). Table 2 shows the averages for these different types of questions and the difference in scores between questions for selecting code segments versus assertions. The latter difference is statistically significant with a p-value of 0.007. While students performed well on all questions when selecting code segments, their scores dropped as they progressed from holistic to formal questions when selecting assertions. Students did worse specifically on the holistic/formal questions when selecting code segments. These observations, paired with the significant dip in students' scores on the formal question when selecting assertions, point to difficulties learning to use mathematical assertions for the first time.

Table 2: Performance on conditional statement questions.

Question	Select Assertion	Select Code	Difference	p-value
Holistic	0.773	0.803	0.030	0.324
Holistic/Formal	0.750	0.682	-0.068	0.183
Formal	0.504	0.682	0.178	**0.007**
Total	0.676	0.722	0.047	0.184

6.3 Comparing Visualization and Post Analysis

We conclude the discussion by connecting visualization analysis with post-test data analysis. Based on qualitative data from the network visualization, we noted in Subsection 6.1 an activity where students in Group A picked the more obviously correct answer in fewer attempts than Group B. Of the students who completed both the lab activities and post-test, those in Group A scored an average of 2.34 out of 4 on the holistic section of the post-test, while students in Group B scored an average of 1.70. These results are summarized in Table 3. Although the difference is not statistically significant, this correlation suggests that the qualitative data on students' thought processes from the network visualization of lab activities may be a predictor of their performance on a post-test.

Table 3: Holistic post-test scores by group.

	Group A	Group B
Count	30	32
Mean (out of 4)	2.34	1.70
Standard Deviation	1.40	1.23

7 SUMMARY

Learning to reason symbolically about the behavior of code involving conditional statements is a difficult task for students. This paper reports on our experience using an online tool designed with suitable activities to communicate the concepts to students. To study student learning, we have used a novel network visualization learning analytics system we built, along with a post-test.

Most students can learn symbolic reasoning through activities. Visualization can highlight not only the clustering of answers and students but also common learning progressions. It may help instructors see that reaching some answers (right or wrong) may be qualitatively better than others and may serve as a prelude to post-test data analysis. Post-test analysis shows that students achieve a measure of holistic or formal thinking. It also shows that students can pick code segments that match assertions better than the opposite, suggesting learning to use assertions for the first time remains a challenge.

Future directions include developing an intelligent tutoring system to provide help where and for whom it is needed and the use of the visualization system in non-CS contexts.

ACKNOWLEDGMENTS

This research is funded in part by US National Science Foundation grants DUE-1914667 and DUE-1915334.

REFERENCES

[1] Olasile Babatunde Adedoyin and Emrah Soykan. 2020. Covid-19 pandemic and online learning: the challenges and opportunities. *Interactive Learning Environments* 0, 0 (2020), 1–13. https://doi.org/10.1080/10494820.2020.1813180

[2] Gökhan Akçapınar, Arif Altun, and Petek Aşkar. 2019. Using learning analytics to develop early-warning system for at-risk students. *International Journal of Educational Technology in Higher Education* 16, 1 (dec 2019), 40. https://doi.org/10.1186/s41239-019-0172-z

[3] Mohammad Alawamleh, Lana Al-Twait, and Gharam Al-Saht. 2020. The effect of online learning on communication between instructors and students during Covid-19 pandemic. *Asian Education and Development Studies* ahead-of-print (08 2020). https://doi.org/10.1108/AEDS-06-2020-0131

[4] Ashok R. Basawapatna and Alexander Repenning. 2010. Cyberspace Meets Brick and Mortar: An Investigation into How Students Engage in Peer to Peer Feedback Using Both Cyberlearning and Physical Infrastructures. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education* (Bilkent, Ankara, Turkey) *(ITiCSE '10)*. Association for Computing Machinery, New York, NY, USA, 184–188. https://doi.org/10.1145/1822090.1822143

[5] Daniel Bauer, Veronika Kopp, and Martin R. Fischer. 2007. Answer changing in multiple choice assessment change that answer when in doubt – and spread the word! *BMC Medical Education* 7, 1 (24 Aug 2007), 28. https://doi.org/10.1186/1472-6920-7-28

[6] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D³ Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (dec 2011), 2301–2309. https://doi.org/10.1109/TVCG.2011.185

[7] William G. Bowen, Matthew M. Chingos, Kelly A. Lack, and Thomas I. Nygren. 2014. Interactive Learning Online at Public Universities: Evidence from a Six-Campus Randomized Trial. *Journal of Policy Analysis and Management* 33, 1 (jan 2014), 94–111. https://doi.org/10.1002/pam.21728

[8] Tracy Camp, W. Richards Adrion, Betsy Bizot, Susan Davidson, Mary Hall, Susanne Hambrusch, Ellen Walker, and Stuart Zweben. 2017. Generation CS: The growth of computer science. *ACM Inroads* 8, 2 (2017), 44–50.

[9] Adam S. Carter, Christopher D. Hundhausen, and Olusola Adesope. 2015. The Normalized Programming State Model. In *Proceedings of the eleventh annual International Conference on International Computing Education Research (ICER '15)*. ACM, New York, NY, USA, 141–150. https://doi.org/10.1145/2787622.2787710

[10] Charles T. Cook, Heather Harton, Hampton Smith, and Murali Sitaraman. 2012. Specification engineering and modular verification using a web-integrated verifying compiler. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, Zurich, Switzerland, 1379–1382. https://doi.org/10.1109/ICSE.2012.6227243

[11] Michelle Cook, Megan Fowler, Jason O. Hallstrom, Joseph E. Hollingsworth, Tim Schwab, Yu-Shan Sun, and Murali Sitaraman. 2018. Where exactly are the difficulties in reasoning logically about code? experimentation with an online system. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2018, Larnaca, Cyprus, July 02-04, 2018*. Association for Computing Machinery, Larnaca, Cyprus, 39–44. https://doi.org/10.1145/3197091.3197133

[12] Svetlana V. Drachova, Jason O. Hallstrom, Joseph E. Hollingsworth, Joan Krone, Richard Pak, and Murali Sitaraman. 2015. Teaching Mathematical Reasoning Principles for Software Correctness and Its Assessment. *TOCE* 15, 3 (2015), 15:1–15:22.

[13] Tommy Färnqvist and Fredrik Heintz. 2016. Competition and feedback through automated assessment in a data structures and algorithms course. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE* 11-13-July (2016), 130–135. https://doi.org/10.1145/2899415.2899454

[14] Louis Faucon, Lukasz Kidzinski, and Pierre Dillenbourg. 2016. Semi-Markov model for simulating MOOC students. In *Proceedings of the 9th International Conference on Educational Data Mining, EDM 2016, Raleigh, North Carolina, USA, June 29 - July 2, 2016*, Tiffany Barnes, Min Chi, and Mingyu Feng (Eds.). International Educational Data Mining Society (IEDMS), Raleigh, North Carolina, USA, 358–363. http://www.educationaldatamining.org/EDM2016/proceedings/paper_112.pdf

[15] Michal Forišek. 2006. On the suitability of programming tasks for automated evaluation. *Informatics in Education-An International Journal* 5, 1 (03 2006), 63–76.

[16] Megan Fowler. 2021. *A Human-Centric System for Symbolic Reasoning About Code*. Ph. D. Dissertation. Clemson University, Clemson, SC 29634.

[17] Megan Fowler, Jason Halltrom, Joseph E. Hollingsworth, Eileen Kraemer, Murali Sitaraman, Yu-Shan Sun, Jiadi Wang, and Gloria Washington. 2021. Tool-Aided Learning of Code Reasoning with Abstraction in the CS Curriculum. *Informatics in Education* 20, 4 (2021), 533–566. https://doi.org/10.15388/infedu.2021.24

[18] Jianxiong Gao, Bei Pang, and Steven S. Lumetta. 2016. Automated feedback framework for introductory programming courses. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE* 11-13-July (2016), 53–58. https://doi.org/10.1145/2899415.2899440

[19] Ginés Gárcia-Mateos and José Luis Fernández-Alemán. 2009. A course on algorithms and data structures using on-line judging. *ACM SIGCSE Bulletin* 41, 3 (2009), 45–49. https://doi.org/10.1145/1595496.1562897

[20] Philippe J. Giabbanelli, Andrew A. Tawfik, and Vishrant K. Gupta. 2019. Learning Analytics to Support Teachers' Assessment of Problem Solving: A Novel Application for Machine Learning and Graph Algorithms. In *Utilizing Learning Analytics to Support Study Success*, Dirk Ifenthaler, Dana-Kristin Mah, and Jane Yin-Kim Yau (Eds.). Springer International Publishing, 175–199. https://doi.org/10.1007/978-3-319-64792-0_11

[21] J. Paul Gibson. 2008. Weaving a Formal Methods Education with Problem-Based Learning. *Communications in Computer and Information Science* 17, 460–472. https://doi.org/10.1007/978-3-540-88479-8_32

[22] David Ginat. 2014. On Inductive Progress in Algorithmic Problem Solving. *Olympiads in Informatics* 8 (2014), 81–91.

[23] Jason O. Hallstrom, Cathy Hochrine, Jacob Sorber, and Murali Sitaraman. 2014. An ACM 2013 exemplar course integrating fundamentals, languages, and software engineering. In *The 45th ACM Technical Symposium on Computer Science Education, SIGCSE 2014, Atlanta, GA, USA, March 5-8, 2014*, J. D. Dougherty, Kris Nagel, Adrienne Decker, and Kurt Eiselt (Eds.). ACM, Atlanta, GA, USA, 211–216. https://doi.org/10.1145/2538862.2538969

[24] Peter B. Henderson. 2003. Mathematical reasoning in software engineering education. *Commun. ACM* 46, 9 (2003), 45–50.

[25] Karin Hess. 2008. Developing and using learning progressions as a schema for measuring progress. *National Center for Assessment* (2008).

[26] Wayne D. Heym, Paolo A. G. Sivilotti, Paolo Bucci, Murali Sitaraman, Kevin Plis, Joseph E. Hollingsworth, Joan Krone, and Nigamanth Sridhar. 2017. Integrating Components, Contracts, and Reasoning in CS Curricula with RESOLVE: Experiences at Multiple Institutions. In *30th IEEE Conference on Software Engineering Education and Training, CSEE&T 2017*. IEEE, Savannah, GA, USA, 202–211.

[27] Sarah Hug, Heather Thiry, and Phyllis Tedford. 2011. Learning to Love Computer Science: Peer Leaders Gain Teaching Skill, Communicative Ability and Content Knowledge in the CS Classroom. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 201–206.

[28] Vladimir Klebanov, Peter Müller, Natarajan Shankar, Gary T. Leavens, Valentin Wüstholz, Eyad Alkassar, Rob Arthan, Derek Bronish, Rod Chapman, Ernie Cohen, Mark Hillebrand, Bart Jacobs, K. Rustan M. Leino, Rosemary Monahan, Frank Piessens, Nadia Polikarpova, Tom Ridge, Jan Smans, Stephan Tobies, Thomas Tuerk, Mattias Ulbrich, and Benjamin Weiß. 2011. The 1st Verified Software Competition: Experience Report. In *FM 2011: Formal Methods*, Michael Butler and Wolfram Schulte (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 154–168.

[29] Victoria Menzies, Catherine Hewitt, Dimitra Kokotsaki, Clare Collyer, and Andy Wiggins. 2016. *Project Based Learning : evaluation report and executive summary.* Project Report. DU, London. http://dro.dur.ac.uk/20513/

[30] Caleb Priester, Yu-Shan Sun, and Murali Sitaraman. 2016. Tool-Assisted Loop Invariant Development and Analysis. In *29th IEEE International Conference on Software Engineering Education and Training, CSEET 2016*. IEEE, Dallas, TX, USA, 66–70.

[31] Kathryn M. Rich, Carla Strickland, T. Andrew Binkowski, Cheryl Moran, and Diana Franklin. 2018. K–8 Learning Trajectories Derived from Research Literature: Sequence, Repetition, Conditionals. *ACM Inroads* 9, 1 (Jan 2018), 46–55.

[32] Beth Simon, Sarah Esper, Leo Porter, and Quintin Cutts. 2013. Student Experience in a Student-Centered Peer Instruction Classroom. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (San Diego, San California, USA) *(ICER '13)*. Association for Computing Machinery, New York, NY, USA, 129–136. https://doi.org/10.1145/2493394.2493407

[33] Murali Sitaraman, Bruce M. Adcock, Jeremy Avigad, Derek Bronish, Paolo Bucci, David Frazier, Harvey M. Friedman, Heather K. Harton, Wayne D. Heym, Jason Kirschenbaum, Joan Krone, Hampton Smith, and Bruce W. Weide. 2011. Building a push-button RESOLVE verifier: Progress and challenges. *Formal Asp. Comput.* 23, 5 (2011), 607–626.

[34] Sotiris Skevoulis and Vladimir Makarov. 2006. Integrating formal methods tools into undergraduate computer science curriculum. In *Proceedings. Frontiers in Education. 36th Annual Conference*. IEEE, IEEE, San Diego, CA, USA, 1–6.

[35] Carol L. Smith, Marianne Wiser, Charles W. Anderson, and Joseph Krajcik. 2006. Implications of Research on Children's Learning for Standards and Assessment: A Proposed Learning Progression for Matter and the Atomic-Molecular Theory. *Measurement: Interdisciplinary Research and Perspectives* 4, 1-2 (2006), 1–98. https://doi.org/10.1080/15366367.2006.9678570

[36] Daniel Spikol, Emanuele Ruffaldi, Giacomo Dabisias, and Mutlu Cukurova. 2018. Supervised machine learning in multimodal learning analytics for estimating success in project-based learning. *Journal of Computer Assisted Learning* 34, 4 (2018), 366–377. https://doi.org/10.1111/jcal.12263

[37] Emily S. Tabanao, Ma. Mercedes T. Rodrigo, and Matthew C. Jadud. 2011. Predicting At-Risk Novice Java Programmers through the Analysis of Online Protocols. In *Proceedings of the Seventh International Workshop on Computing Education Research* (Providence, Rhode Island, USA) *(ICER '11)*. Association for Computing Machinery, New York, NY, USA, 85–92. https://doi.org/10.1145/2016911.2016930

[38] Daan Vermaak. 2019. *Modeling, Visualizing, and Analyzing Student Progress on Learning Maps*. Ph. D. Dissertation. University of Kansas.

[39] Camilo Vieira, Paul Parsons, and Vetria Byrd. 2018. Visual learning analytics of educational data: A systematic literature review and research agenda. *Computers & Education* 122 (July 2018), 119–135.