AERA Open
January-December 2021, Vol. 7, No. 1, pp. 1–25
DOI: 10.1177/23328584211008111
Article reuse guidelines: sagepub.com/journals-permissions

An Ethnomethodological Study of Abductive Reasoning While Tinkering

ChanMin Kim
Brian R. Belland
Afaf Baabdullah
Eunseo Lee
Emre Dinç
Anna Y. Zhang
The Pennsylvania State University

Tinkering is often viewed as arbitrary practice that should be avoided. However, tinkering can be performed as part of a sound reasoning process. In this ethnomethodological study, we investigated tinkering as a reasoning process that construes logical inferences. This is a new asset-based approach that can be applied in computer science education. We analyzed artifact-based interviews, video observations, reflections, and scaffolding entries from three pairs of early childhood teacher candidates to document how they engaged in reasoning while tinkering. Abductive reasoning observed during tinkering is discussed in detail.

Keywords: *teacher learning, equitable computer science education, robot programming and debugging, tinkering, abductive reasoning*

Tinkering can be defined as playful experimentation that involves a series of changes incrementally applied without systematic planning (Beckwith et al., 2006; Berland et al., 2013; Blikstein et al., 2014). To improve computer science education for learners from underrepresented populations in STEM (science, technology, engineering, and mathematics), it is important to acknowledge tinkering as a productive and even necessary process (Berland et al., 2013; Searle et al., 2014). Since the seminal work by Turkle and Papert (1990), discussions on tinkering have mainly focused on tinkering outcome quality, especially as compared with those of structured programming (e.g., Rose, 2016). But little is known about *actions of reasoning* within tinkering.

Given this gap, we examined conversations and actions of early childhood teacher candidates during debugging to understand their reasoning while tinkering. The research question was, "How do early childhood education (ECE) teacher candidates reason while tinkering?" Because we assume that such reasoning is not an additive process of two individuals' cognitions, but rather a synergistic product of interaction, we used an ethnomethodological approach (Belland et al., 2016; Garfinkel, 1967; Ingram, 2018) to analyze the data. Specifically, we sought to uncover the methods by which teammates sought to achieve order within their interactions directed toward debugging. From this analysis, we uncovered rules that guided tinkering—rules that were endemic to interaction. The focus of conversation analysis within the ethnomethodological approach was on tinkering "actions rather than the content of talk itself" (Ingram, 2018, p. 1068). This was not only to understand how the participants made sense of bugs and debugging but also to identify particular "features of the interaction that precede particular outcomes" from tinkering (Ingram, 2018, p. 1069).

Relevant Literature

Teacher Learning of Debugging

Few ECE teachers are prepared to teach computer science in the United States and globally (Bers et al., 2013; Çetin & Demircan, 2020). Within ECE, block-based programming platforms are often used (Bers, 2018; Bers et al., 2014; Kim et al., 2015; Kim et al., 2017; Kim et al., 2020). While research shows that novice learners are more attracted to block-based programming than text-based programming (Akcaoglu, 2014; Bers, 2018; Bers et al., 2014; Lye & Koh, 2014), actual learning of fundamental computer science concepts and programming through block-based platforms has been criticized (Brennan & Resnick, 2012; Grover et al., 2015). Still, learning to debug could improve incomplete understanding of programming (Kim et al., 2018). For example, when debugging to make a robot travel on a square, sequence and repetition control structures need to be understood to change 3 to 4 in the repeat block (turquoise in Figure 1).

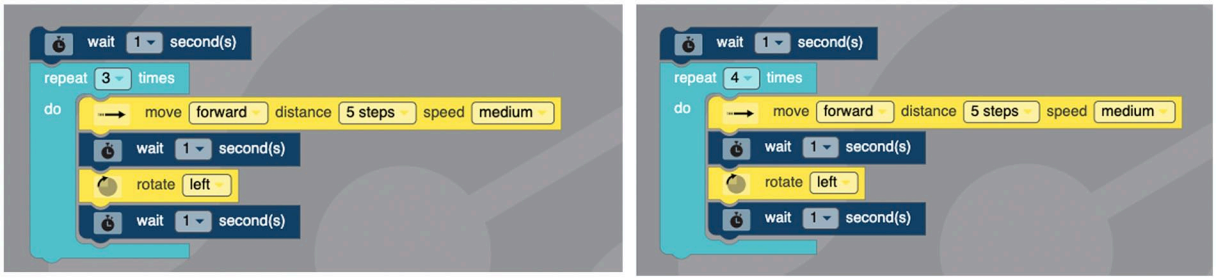


FIGURE 1. *Block-based programing examples of buggy code (left) and correct code (right).*

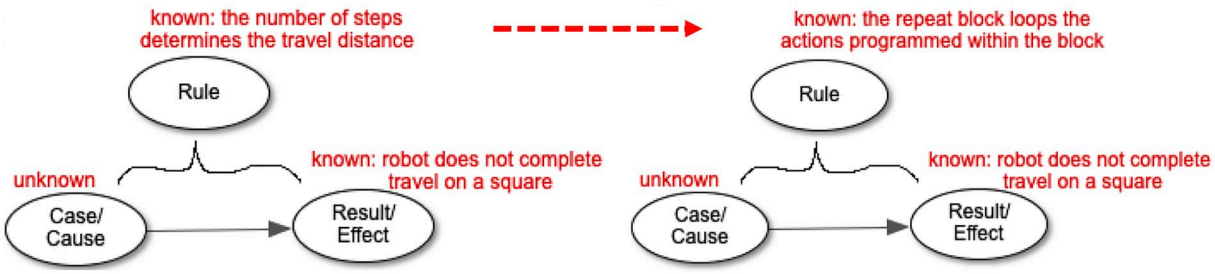


FIGURE 2. *Simplified illustration of reasoning while tinkering during debugging.*

Tinkering as an Asset

In computer science education, it has been long argued that tinkering is a culture of programming that can lead to positive outcomes (Berland et al., 2013; Rose, 2016; Turkle & Papert, 1992). Still, tinkering is often viewed as a practice to be avoided (Grigoreanu et al., 2006; Murphy et al., 2008). In a recent study, an intervention structured the debugging process to help students avoid "rapid cycles of editing and testing without deeply understanding their code" (Ko et al., 2019, p. 474). Others acknowledge that *some* forms of tinkering are productive, especially those involving some questioning and hypothesizing (Beckwith et al., 2006; Fitzgerald et al., 2010; Perkins et al., 1986).

We argue that tinkering during debugging involves *thinking* to trace the cause that leads to the result of the code. For example, to fix the code in Figure 1, learners run the robot and investigate why it does not complete a square. They do not know yet that the value, 3, in the repeat block should be corrected. They simply think that the distance the robot traveled is too short to complete the square and thus increase the number of steps in the move block. A rule applied in reasoning here for what caused the result is that the number of steps determines the travel distance. The rule was not the right one for this debugging task, and the change does not debug the code. They run the robot again to see how much further it should travel. They now realize that it travels on only 3 sides. They attend to the repeat block and enter 4 instead of 3. A rule applied here is that the repeat block loops the action programmed inside the block the number of times entered in the block. Figure 2 depicts this possible reasoning while tinkering during debugging. It should be noted that this figure illustrates tinkering in a simplified way. Novice programming learners are unlikely to have a vast repertoire of relevant rules that can be applied in debugging (Metzger, 2003), and actual tinkering processes may involve extended rounds of applying irrelevant rules before applying a relevant rule.

Reasoning Types

The debugging example above depicts abductive reasoning, defined as a reasoning process used to explain a phenomenon with incomplete information at hand (Abe, 2003; Aliseda, 2006; Leake, 1995; Magnani, 2009, 2015; Peirce, 1931-1935). Through abduction, one reasons probable cases/causes that seem to have resulted in the phenomenon. The most probable case at the moment is tested to see if it explains the phenomenon. In the example above, the first probable case (i.e., the bug is in the move block) does not explain the robot's failure to complete the square, which leads to the second probable case (i.e., the bug is in the repeat block). Other types of reasoning are often used in programming. Deductive reasoning and inductive reasoning are mainly discussed in

Abduction (Figure 3a)
When learners know:
the rule: the repeat block loops the action programmed inside of the block n times
the result: the action is repeated 4 times
they can abduce:
the case: n = 4 in the repeat block

Induction (Figure 3b)
When learners know:
the case: n is entered in the repeat block
the result: the action is repeated n times
they can induce:
the rule: perhaps the repeat block loops the action programmed inside of the block n times

Deduction (Figure 3c)
When learners know:
the rule: the repeat block loops the action programmed inside of the block n times
the case: n = 4 in the repeat block
they can deduce:
the result: the action programmed inside of the repeat block will be repeated 4 times

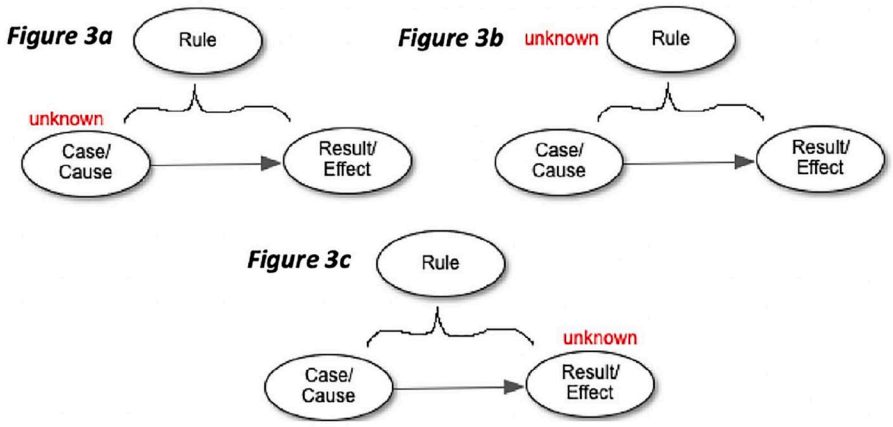


FIGURE 3. *Simplified illustration of reasoning types.*

the debugging literature (e.g., Dershowitz & Lee, 1993; Metzger, 2003; Zeller, 2009). The three types of reasoning are explained in Figure 3 with a task of coding a robot to repeat an action *n* times. Peirce's (1931-1935) illustration of abduction and Abrahamson's (2012) inference mechanism example were the basis for constructing these examples.

Inductive debugging involves enumerating all cases and studying their relationships with the results. To induce from specific cases to general rules, cases of code that function correctly and incorrectly should be reviewed (Myers et al., 2004). To use deduction in debugging, a rich body of general knowledge drawn on extensive programming experience as well as a full understanding of the current program is necessary (Metzger, 2003). Deduction is commonly considered a strong method of debugging (Myers et al., 2004; Zeller, 2009). However, deductive debugging is usually not recommended to novice programmers (Metzger, 2003). This appears to leave a large role for abduction in the study of novices' debugging. Debugging by abduction is scarcely discussed outside machine learning development. To our knowledge, the present research is the first that analyzed abductive debugging in a block-based coding context.

Method

Participants and Setting

The setting was an undergraduate class on play in early childhood education at a large, public university in the eastern United States. Out of 19 who consented to participate in the overall study, we focused on three pairs of participants in this qualitative case study (all female; see Table 1). All three pairs self-reported having no to little prior programming knowledge and no prior robot programming experience. Amelia, Barbara, and Charlotte reported having low prior programming knowledge on a closed-ended item in a presurvey. In reviewing other data (i.e., classroom recordings, interviews, reflections, scaffolding entries), we found no detail about, or evidence of the participants referring to, such prior programming knowledge. Two of the three pairs were homogeneous in academic standing.

Robot Programming Unit

Participants worked in pairs to explore how to (a) integrate block-based coding into children's play and learning and (b) debug errors that impeded successful execution of block-based code. For example, participants were asked to debug code that had two bugs (in the repeat while and rotate blocks) that impeded the Ozobot from tracing an octagon shape (Figure 4). The unit lasted 3 weeks with a 2.5-hour class each week (see Table 2). Participants used OzoBlockly to instruct Ozobot Evo to make desired movements. OzoBlockly is a block-based programming platform with five levels to support coding from prereader to advanced learners.

Data Collection

Classroom Recordings. The three pairs were video recorded for about 6 hours in three classes. The video recordings were transcribed by advanced speech recognition software. Two researchers edited the transcripts to improve accuracy.

Interviews. Individual interviews were done after the unit (mean interview length = 28.3 minutes). Participants' responses to the scaffold were used to prompt recollection. Participants were also asked to debug code that instructed an Ozobot to trace a hexagon.

Reflections. Participants wrote (a) how they might support children's learning in play, (b) how their play scenario was designed with Ozobots, (c) how they engaged in debugging, (d) how they would apply their debugging experience to children's learning, and (e) how they would support children's problem-solving.

Scaffolding Entries. Debugging scaffolding was provided for listing block categories, generating and testing hypotheses, and reflecting on hypotheses. Specifically, prompts invited students to (a) list block categories in the buggy code; (b) generate two or three hypotheses along with their reasons; (c) record changes they made in the code, the reason for those changes, and the change outcomes for each hypothesis, and the list of blocks after changes were made; (d) summarize and reflect on the hypotheses they tested by writing down the actual problem in the buggy code, how they fixed the problem, and whether any of the hypotheses was correct (if not, they were encouraged to generate another hypothesis); and (e) list block categories used in the final code. The scaffold included an example hypothesis: "The numeric value that I entered in the Loops block was too small. It should have been 4 instead of 3 for my Ozobot to get to the finish line."

Data Analysis

The initial coding scheme was developed based on the programming, debugging, and problem-solving reasoning literatures (Abrahamson, 2012; Aliseda, 2006; Berland et al., 2013; Kim et al., 2018; Magnani, 2015; McCauley et al., 2008; Perkins et al., 1986; Rivera & Becker, 2007; Turkle & Papert, 1990). To refine the initial coding scheme, two researchers conducted preliminary coding of data of two participants from different video-recorded pairs. Then, the project team discussed the coded data and revised the coding scheme. Next, five researchers coded the two participants' data independently and peer-reviewed each other's coded data. The project team met again to discuss the recoded data and finalized the coding scheme. Using the refined coding scheme, three researchers independently coded the transcript of videos of the two participants (Cohen's kappa ranged from .84 to .87) and then met to discuss their analysis with another researcher in order to reach consensus. Next, three researchers coded the remaining data and went through multiple rounds of peer reviews and revisions with another researcher. After finishing coding, the three researchers independently created a sensemaking table of their coded data to find relations among coded data under different nodes and to identify salient observations. Four researchers then (a) peer-reviewed, discussed, and revised the sensemaking tables and (b) repeated the process to finalize the sensemaking tables. Throughout this process, one of the key sticking points was how to use the coding node *tacit construction*, defined as applying preexisting perceptual and reasoning tools to model a problem space (Abrahamson, 2012). The sticking point was that, when coding the data, we needed to consider whether participants were using disciplinary tools or their prior or concurrent perceptual experiences, when constructing the problem space. This required us to consult multiple data sources and to carefully use the lens of ethnomethodology as we considered participants' actions, as these can often most appropriately be viewed as the synergistic product of

TABLE 1
Participant Demographic Information

Participant pseudonym	Team No.	Age in years	Academic standing	Self-reported prior programming knowledge	Self-reported prior robot programming experience
Charlotte	1	20	Sophomore	Low knowledge	No experience
Harper	1	20	Sophomore	No knowledge	No experience
Barbara	2	20	Junior	Low knowledge	No experience
Lillian	2	21	Junior	No knowledge	No experience
Amelia	3	20	Sophomore	Low knowledge	No experience
Elena	3	21	Senior	No knowledge	No experience

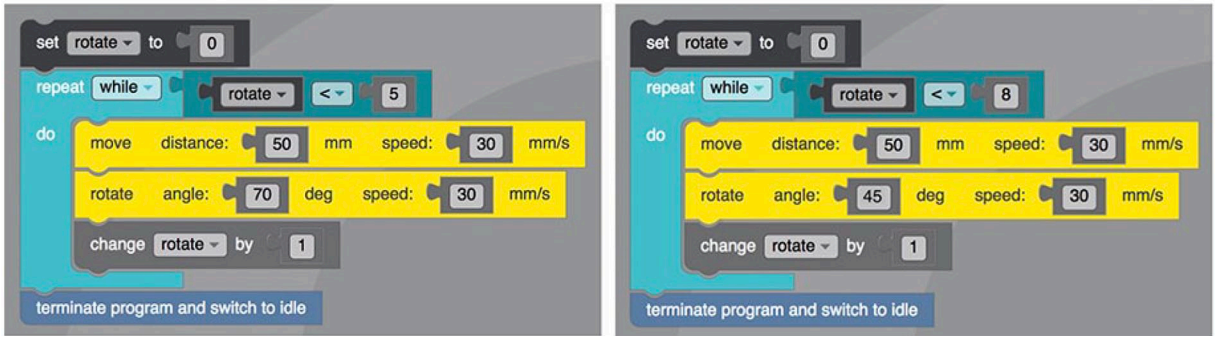


FIGURE 4. *Buggy code (left) and an example of correct code (right) for the octagon tracing task.*

TABLE 2
A Summary of the Unit Activities

Week 1	Week 2	Week 3
• Introduction to robots and their use in education • Introduction to Ozobot Evo and OzoBlockly • Practice of calibration, loading code on Ozobots, and running Ozobots • Introduction to coding in OzoBlockly Level 3 • Reflection on how Ozobots could be used in children's play and learning	• Showcase of OzoBlockly code that participants created as homework during Week 1 • Introduction to coding and debugging, Level 3 line navigation • Debugging with scaffolding • Reflection on the debugging task • Reflection on children's play • Exploration of OzoBlockly Level 4	• Showcase of OzoBlockly code that participants created as homework during Week 2 • Debugging practice • Debugging with scaffolding • Reflection on children's play involving debugging

pairs' joint cognition (Garfinkel, 1967). NVivo 12 was used for analysis.

Findings and Discussion

Abduction was often observed in debugging episodes in that participants (a) determined plausible cases (causes) that explained misfunctioning code (the result) (b) by applying the best rule at the moment that connected the case to the result, and (c) eliminated irrelevant rules when realizing a disconnect between the case and the result through seeing reprogrammed robot behaviors. This process of abduction seemed to have necessitated tinkering. Tinkering observed in the study was also rather cautious (Berland et al., 2013; Perkins et al., 1986). Considering that sociocognitive processes can mediate abductive reasoning (Abrahamson, 2012), scaffolded pair debugging seemed to have guided participants to track code, defined as reading code closely to get a sense of what it does (Perkins et al., 1986). During interviews, participants noted that responding to scaffolding prompts made them look back at their thinking process and the results (i.e., reprogrammed robot behaviors) of their

actions (i.e., code revision) taken based on the thinking process.

Our findings can be summarized by the following themes:

Theme 1: Tacit rules were applied in search for best explanations.
Theme 2: Rules were eliminated based on perpetual observations.
Theme 3: Reflective abstraction was seldom observed.
Theme 4: Generating multiple hypotheses in advance was an unnatural requirement.

The themes are sequenced as such because each subsequent theme builds off of the previous theme. Our presentation of Theme 1 reports the process of abduction as a whole in which tacit rules were used to abduce the case (code) that led to the result (robot behavior). When presenting Theme 2, we report perceptual observations that were immediately available during robot debugging, and how such perceptual observations contributed to rule elimination during abduction. In our presentation of Theme 3, we report that scaffolding largely served to facilitate reflective debugging rather than reflective abstraction. Our reporting of Theme 4 centers on participants' perceptions of the scaffolding requirement to generate multiple hypotheses in advance and reports how they used the scaffolding.

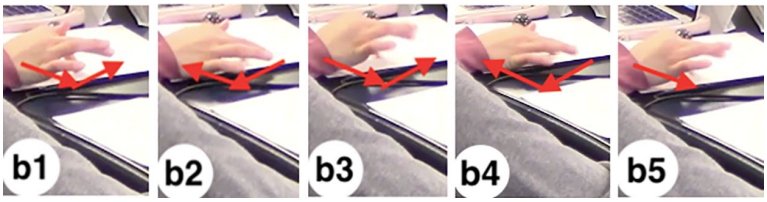
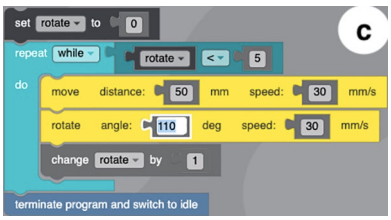
Theme 1: Tacit Rules Were Applied in Search for Best Explanations

Use of Tacit Rules in Abducing the Case Leading to the Result. The rules participants used in search of a best explanation for the malfunctioning code were rarely articulated. For example, when Charlotte and Harper talked about the angle value, 70, in the rotate block (see Figure 4), they did not state a rule that made them think that the angle was too big. See Table 3 for the dialogue of one debugging episode in which they changed (a) the angle value from 70, to 110, 50, and 45 in the rotate block and (b) the number in the repeat while block from 5 to 8. The episode is segmented into a series of scenes when a new change in the buggy code was mentioned. In the reasoning analysis column, the *result* is what the pair saw *and* noticed during the scene. For example, Scene 1 and Scene 2 illustrate the pair's conversation about why 70° in the rotate block causes the robot to veer off the octagon. But in Scene 1, the pair talked about the robot's (too-wide) rotation, and in Scene 2, the pair talked about the robot's (too-sharp) turn during its transition between the sides of the octagon. That is, what they see is the same but what they attend to and talk about is different in the two scenes. This is why the results identified in this column can differ in scenes with the same code and/or the same robot behavior. The *case* is what the pair concluded to be the cause of the problem. In Scene 1, the pair talked about replacing 70° with 50°, and in Scene 2, the pair talked about replacing 70° with 110°. Such code revision plans are to fix the causes (i.e., cases) of the problem. Thus, in Scene 1, the case is "70° is too big," and in Scene 2, the case is "70° is too small." The *rule* is what the pair used to justify their claim about the case regardless of its accuracy and/or applicability. In Scene 1, Charlotte explained that 70° should be replaced with 50° because the robot rotated too widely. The rule inferred in this scene is about the relationship between rotation size and the angle value (i.e., "when the robot rotates too widely, the angle value in the rotate block is too large"). In Scene 2, Harper explained that 70° should be replaced with 110° because the robot turned too sharply. The rule inferred in this scene was about the interior angle of an octagon (e.g., "when the robot traces an octagon, its transition between consecutive sides should be as wide as the interior angle of an octagon"). Rules were tacit rather than explicitly spoken. Figure 5 illustrates the tacit rule used to figure out the unknown case resulting in too-wide rotations in Scene 1. The pair abduced the case (i.e., 70° is too big) that led to the result (i.e., robot rotation angle is too wide to trace the octagon). The tacit rule that they used to make sense of the result is that the size of the robot rotation angle is determined by the angle value in the rotate block.

Role of Perceptual Senses in Using Tacit Rules. Without stating the rule, the pair looked at and traced the robot movement and the octagon on the paper. Instead of trying to describe or justify the rule that is being applied, they seemed to use their perceptual senses. This makes sense in that the rule could be checked immediately by revising the code. Through changes made in the code, they collected "perceptual facts" and dialogued about "the evidence of the senses" with each other (Peirce, 1931-1935, para. 141). The use of tacit rules seems necessary in abductive reasoning considering that while "discovering through doing . . . new and still unexpressed information is codified by means of manipulations of some external objects" (Magnani, 2009, p. 11). Abductive inferences and perceptual judgments go together (Hanson, 1958).

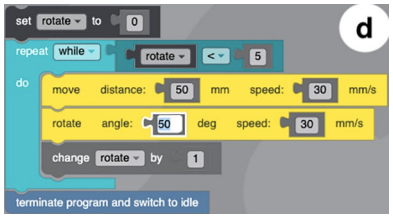
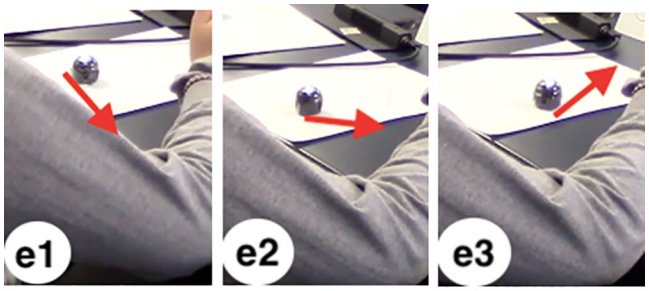
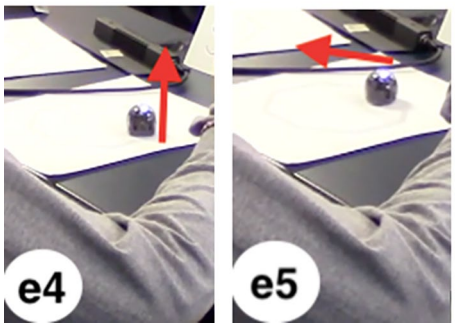
Some "naïve perceptual judgment" (Abrahamson, 2012, p. 636) led to unsound explanations that had to be eliminated later. This is a natural part of abduction and the reason why abductive reasoning was needed and applicable. Unlike deductive inferences, abductive inferences can work with "an incomplete knowledge base" (Abe, 2003, p. 233). For example, knowing that $A \rightarrow B$ *and* $B \rightarrow C$, one can conclude $A \rightarrow C$ through deduction, but if either piece of information is missing, deduction cannot be done but abduction can. Discovery (Abe, 2003), creative problem-solving, and design (Kolko, 2009) are possible by abduction despite incompleteness, constraints, or uncertainty. Debugging tasks were *given* to participants of the present study. Debugging

TABLE 3
Charlotte and Harper's Octagon Debugging Episode

Scenes	The pair converses . . .	It is inferred that . . .
Scene 1	Harper [10:15]: I think it's the, okay. But I think it's this rotation [angle] needs to be less. Charlotte [10:23]: Yes, because it's rotating *(She pointed at the point of the octagon shape.)* ⓐ  Harper [10:26]: So we need to rotate it [Ozobot]. Like what do you think? Like fifty? Charlotte [10:30]: Yeah, I'll try 50.	With: *the result*: with 70° in the rotate block, the robot rotates too widely. *the rule*: the angle value in the rotate block determines the size of the rotation that the robot makes; when the robot rotates too widely, the angle value in the rotate block is too large. the pair abduced: *the case*: 70° is too big.
Scene 2	Harper [10:33]: well ⓑ *(She traced the angle between two consecutive sides of the traced octagon with her finger.)*  Charlotte [10:37]: It's like . . . Harper [10:38]: No, no, no, no. The opposite, [the angle] has to get bigger because 50-degree angles like that [inaudible] has to be obtuse like up to 110 or something. 'Cause that's not already 90 degrees. *(She visualized an acute angle with her fingers in the air, and then visualized an obtuse angle with her hand on the paper like she did in ⓑ)* Charlotte [10:50]: It's not 90 degrees, yeah. Harper [10:53]: Yeah, like maybe like 110. ⓒ  Charlotte [10:55]: Okay. Harper [10:56]: Let's see. It's like that at least works just for now and then. Charlotte [10:59]: Yeah *(She loaded the code with 110 degrees and ran the Ozobot on the octagon shape. The Ozobot followed one side, turned to the inside of the octagon, and did not follow the octagon shape.)*	With: *the result*: with 70° in the rotate block, the robot veers off of the traced octagon by turning too sharply during its transition between consecutive sides. *the rule*: when the robot traces an octagon, its transition between consecutive sides should be as wide as the interior angle of an octagon. the pair abduced: *the case*: 70° is too small.

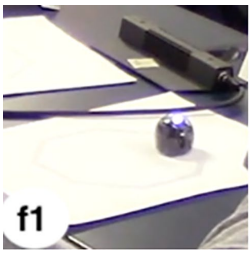
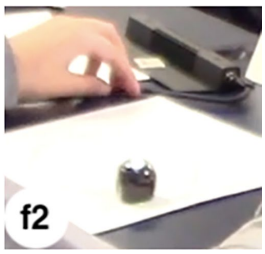
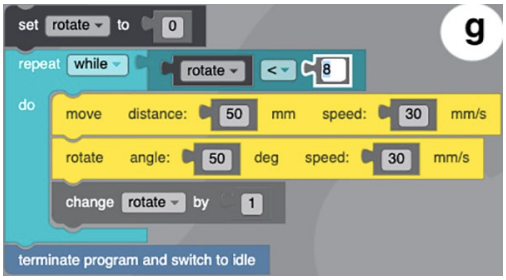
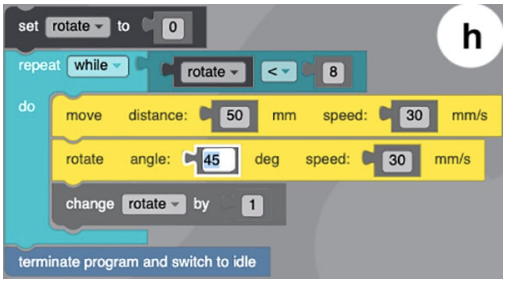
(continued)

TABLE 3 (CONTINUED)

Scenes	The pair converses . . .	It is inferred that . . .
Scene 3	Harper [11:28]: *(After seeing the Ozobot veering off of the traced octagon with 110° in the rotate block)* Okay. Now it's like maybe we had to do less than. Charlotte [11:39]: Alright, let's then try 50. Ok? ⓓ  *(Charlotte calibrated the Ozobot and loaded the revised code onto the Ozobot)* Harper [11:51]: I know we have to really do it [calibration] every single time [we change the code]. Charlotte [12:12]: I hate that [calibration] thing. Harper [12:39]: Yes. Okay. So that's where it [Ozobot] goes *(Charlotte ran the Ozobot on the octagon shape. The Ozobot travelled on the shape quite closely with a little deviation toward the inside of the octagon.*ⓔ*)*   Charlotte [12:45]: Generally, it's really close.	With: *the result*: with 110° in the rotate block, the robot veers off of the traced octagon by rotating even more widely than with 70° in the rotate block during its transition between the consecutive sides. *the rule*: the angle value in the rotate block determines the size of rotation that the robot makes; when the robot rotates too widely, the angle value in the rotate block is too large. the pair abduced: *the case*: 70° is too big.
Scene 4	Harper [12:48]: Oh wait. Why does it [Ozobot] stop? *{Harper and Charlotte expected that the Ozobot would travel around the octagon shape by changing the angle}*. Is your thing [Ozobot] like dead or something? Charlotte [12:50]: Maybe because of that [mic cable]ⓕ *(The Ozobot stopped next to the mic cable. She ran the Ozobot again while holding the mic cable above the paper.)*	With: *the result*: with 5 in the repeat while rotate < block, the robot does not complete its travel on 8 sides of the octagon.

(continued)

TABLE 3 (CONTINUED)

Scenes	The pair converses . . .	It is inferred that . . .
	  Harper [13:09]: Okay. But see that it [Ozobot] literally goes five times, so that's why I want to say. Charlotte [13:14]: A number this *(She changed the number for the repetition count from 5 to 8 without saying so out loud.)*ⓖ  Harper [13:18]: maybe with the change of angle [the rotations] will change *{She noticed the bug, the number of rotations, but thought the number depends on the angle.}*	*the rule*: the number in the rotate < block should be equal to the number of sides of the octagon. the pair abduced: *the case*: 5 is too small.
Scene 5	Charlotte [13:30]: I don't really know how to work this fully. *(She loaded the code.)* Harper [14:05]: It's like so close. I think maybe what should we do like, um, or would it be . . . Charlotte [14:13]: 55, 30, 45? Harper [14:18]: 45, I think. Charlotte [14:21]: I am gonna check off *(She changed the angle from 50 to 45.)*ⓗ  Harper [14:32]: That's where it felt like it [Ozobot] didn't do 8 times until we change the angle. *{She explained her observation of the relationship between number of rotations and angle.}* Charlotte [14:35]: I know. Harper [14:38]: Wonder why. *(Charlotte loaded the code.)* *(Charlotte ran the Ozobot with 8 repetition count and 45 degrees angle on the octagon shape. The Ozobot behaved as desired by moving around the octagon shape but with a little deviation of the angle that resulted from the Ozobot's starting position.)* Charlotte [15:06]: That's like pretty close. I'd say that's good enough.	With: *the result*: with 50° in the rotate block, the robot still veers off of the traced octagon by rotating too much during its transition between the consecutive sides. *the rule*: the angle value in the rotate block determines the size of rotation that the robot makes; when the robot rotates too widely, the angle value in the rotate block is too large. the pair abduced: *the case*: 50° was too big.

Note. Digits within brackets are time stamps. Text within brackets represents (a) missing word(s). Text in parentheses is the researchers' description of the participant's actions or the robot's movement that occurred at the same time as the transcribed speech in the immediately preceding sentence. Text within curly brackets is the researchers' description of the function of the dialogue and/or action.

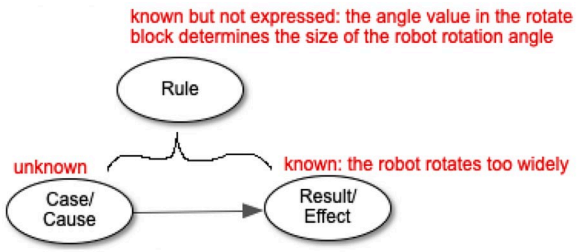


FIGURE 5. *An example of abductive reasoning looking for the best explanation for too-wide rotations.*

code created by somebody else creates more difficulties, external constraints, and uncertainty even with a description of the programming goal (McCauley et al., 2008). In the case of the octagon debugging task, even the size and regularity of the octagon were set, which allowed less flexibility in revising the buggy code.

Tacit rules used when formulating unsound explanations were also based on perceptual senses. Some rules were not relevant or conflicted with each other but later were eliminated in the process of tinkering by abduction. For instance, as illustrated in Scene 2 of the debugging episode in Table 3, Harper disapproved of her partner's and her own previously reasoned case (see Scene 1 in Table 3). This time, she paid more attention to the octagon that the robot should *trace* than the robot itself that should *rotate*. The robot needed to rotate at a 45° angle during a transition between consecutive lines and then keep going straight, but she seemed to think of this movement as merely following the lines making an obtuse angle. Her newer perceptual judgment caused her to abandon the previously used rule and generate a new tacit rule: When the robot traces an octagon, its travel between consecutive sides should be as wide as the interior angle of an octagon. Grounded in this rule, Harper said "[the angle value] has to get bigger." In Scene 2, the pair thought that 70° was too small (the case) because the robot veered off the traced octagon during its transition between consecutive lines (the result). The rule was unsound and fell into disuse immediately after trying out a bigger angle than 70° (110°) in the rotate block.

In Scene 3, Charlotte and Harper went back to the initial rule used in Scene 1 and the initial explanation: 70° was too big. Then, they tried 50°. They did not converse about another applicable rule: The rotate angle value should be equal to the exterior angle of a regular octagon ($\therefore 360°/8 = 45°$). It is probable that the pair knew the rule about exterior and interior angles in an octagon, but they were too occupied with unfamiliar programming activities to notice the relevance of the rule to the present debugging task. If this rule were used, the tinkering process would have been shorter: One change from 70° to 45° could have been made instead of changes from 70° to 110°, 50°, and 45°.

Theme 2: Rules Were Eliminated Based on Perceptual Observations

Rule Elimination Grounded in Ongoing Observation. Abductive reasoning involves eliminating rules, and it is often done with incomplete information. Rule eliminations in the present study were mostly based on ongoing observations while *doing*, often without incorporating prior experience of programming or a general rule. In most episodes, this process of rule eliminations by *doing* led to successful debugging especially on the octagon task. For example, as described above, the rule used by Charlotte and Harper in Scene 2 was eliminated when changing the value to 110° in the rotate block did not fix the bug (see Table 3), and the pair eventually corrected the angle to 45°. However, when observations missed critical details or were simply inaccurate, relevant rules were eliminated because the elimination decision was dependent on observed outcomes of *doing*. For example, as shown in Table 4, the pair changed the numeric value in the compare block within the repeat while block from 5 to 8 (correctly), but they miscounted the number of sides the robot traveled on while observing the change outcome and concluded (incorrectly) that the change did not fix the bug. The incorrect angle value in the rotate block (i.e., the other bug) made the tracing and counting difficult.

This erroneous elimination of a relevant rule seems to have stemmed from *little sensemaking while* and *after doing*. When they changed the numerical value back to 5 in the repeat while block, Charlotte and Harper conversed to make sense of the buggy code. However, their attempt for *sensemaking after doing* their prior change to 8 was not completed. Sensemaking became more complicated when considering the last block in the loop: Change rotate by 1 (i.e., change the variable "rotate" that was set as 0 in the beginning of the code). Since this first debugging attempt, Charlotte and Harper tinkered through to successful debugging as listed in Table 3. While they recognized the interrelatedness between the number of rotations and the turn angle during tinkering, they did not make sense of why 8 did not work during their initial debugging attempt (see Scene 5 in Table 3). This is a natural process within abductive reasoning, which involves "doing-before-understanding" by nature (Abrahamson, 2012, p. 630), and even more so in manipulative abduction (Magnani, 2009; Park, 2017) that relies on perceptual facts that are imperfect (Peirce, 1931-1935).

Sensemaking While Doing. Barbara and Lillian eliminated rules mostly based on observations while *doing* as observed during Charlotte and Harper's debugging. For example, as shown in Scene 1 of Table 5, Barbara and Lillian decided to try an obtuse angle for the same reason that Charlotte and

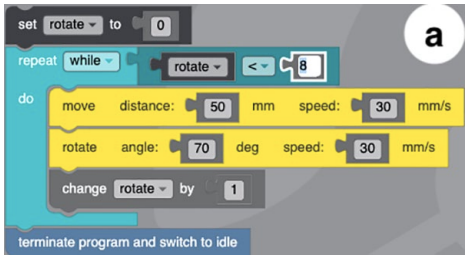
TABLE 4

Charlotte and Harper's Octagon Debugging Initial Episode

The pair converses . . .

Charlotte [05:48]: We have to change it [the code] first because we have to fix it [the code], right? So, maybe this number? *(She pointed at the number of rotations in the code with her mouse cursor.)* I don't know, that's 5. We said it [Ozobot] was moving, moving that 8, I don't know.

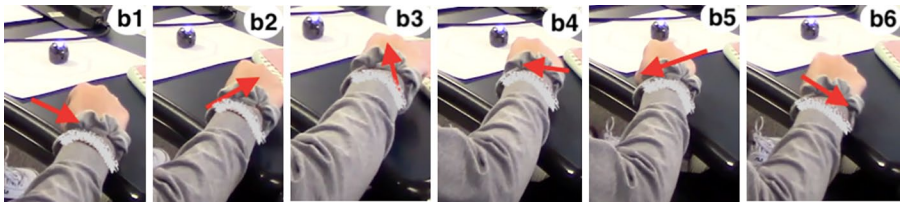
Harper [06:02]: Yeah, yeah. Try that and try that to 8. ⓐ *(Charlotte changed the number of rotations from 5 to 8.)*



Charlotte [06:26]: Okay. Let's see. *(She loaded the revised code onto the Ozobot and ran the Ozobot.)*

Harper [06:40]: It doesn't look like it changed. *(She stated this while looking at the Ozobot's movement.) {She reported that the revised code did not correct the Ozobot's movement.}*

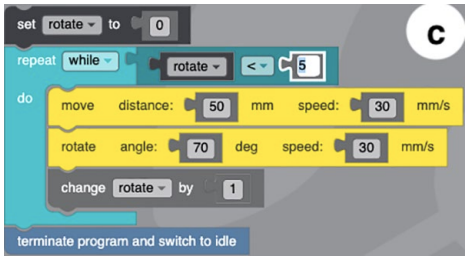
Charlotte [06:42]: No. *(She agreed with Harper.)* It [Ozobot] is making a . . . 'cause it's going . . . What is that? One, two, three . . . one, two, three, four, five *(She traced the movement of the Ozobot with her finger on the table.* ⓑ*)*. I think it's making . . . it was making five . . . a pentagon? *{She explained that the Ozobot traveled on five sides.}*



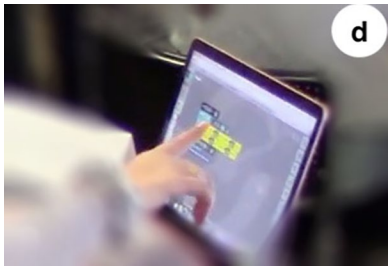
Harper [07:00]: Yeah, pentagon, okay, so that didn't do anything. *{She explained that the change from 5 to 8 in the repeat while block had no impact on the Ozobot's movement.}*

Charlotte [07:04]: Should I put it back to 5?

Harper [07:05]: Yeah, I guess. *{Harper and Charlotte unnoticed that the change to 8 did fix one of the bugs so they changed 8 back to 5.* ⓒ*}*



Charlotte [07:21]: I don't understand what that means "repeat while rotate greater than" like what is that*? (She pointed at the repeat while block.* ⓓ *Please note that she misread < as greater than.)*



(continued)

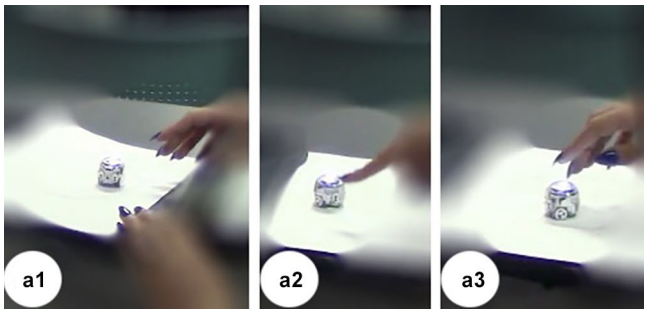
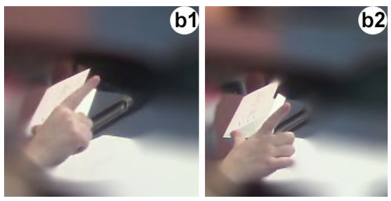
TABLE 4 (CONTINUED)

The pair converses . . .
Harper [07:29]: Repeat while *(She read the repeat while block.)* Charlotte [07:32]: Set rotate to right now set rotate to zero (*She read the set rotate to block.)* Harper [07:38]: It's rotating. *(She explained that the set rotate to block was to make the Ozobot rotate.)* It's just, this one's tricky. Charlotte [07:49]: You don't think we need more of these, right? Move. Rotate. *(She pointed at the move blocks.)* Harper [07:58]: Maybe if we add two more, maybe there's . . . *{She noticed that the code needed more move and rotate blocks.}* Charlotte [08:03]: Moves, rotates, change rotate by one and it moves. *(She traced the movement of Ozobot on the table with her finger.* ⓔ*)*  Harper [08:13]: Move, rotate. Charlotte [08:18]: Wait, I think I can do it again. *(She re-ran the Ozobot.)* Moves, rotates *(She stated this while looking at the Ozobot's movement on the paper.)* Harper [08:33]: Moves, rotates, moves, rotates, moves, rotates. *(She stated this while looking at the Ozobot's movement on the paper.)* Charlotte [08:42]: What does this "change rotate by one?" *(She pointed at the change rotate by 1 block with her mouse cursor.)* Like what does that do?

Note. Digits within brackets are time stamps. Text within brackets represents (a) missing word(s). Text in parentheses is the researchers' description of the participant's actions or the robot's movement that occurred at the same time as the transcribed speech in the immediately preceding sentence. Text within curly brackets is the researchers' description of the function of the dialogue and/or action.

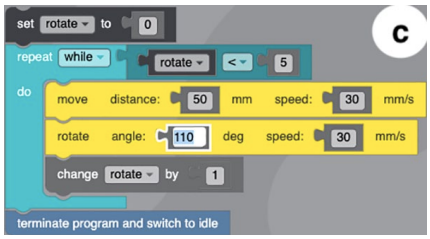
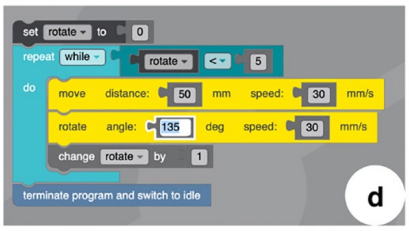
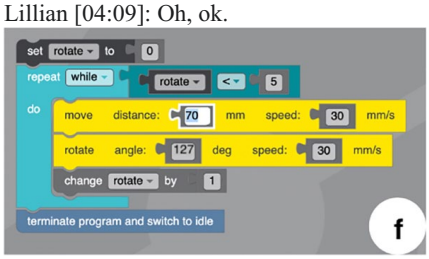
TABLE 5

Barbara and Lillian's Octagon Debugging Episode

Scenes	The pair converses . . .
Scene 1	*(Barbara repositioned the Ozobot on the shape every time it deviated from the shape.* ⓐ*)*  Lillian [01:09]: So it's too small of an angle. So 90 degrees is this, so it's over 90. *(She showed the right angle* **b1** *and the obtuse angle* **b2** *with her finger.)* 

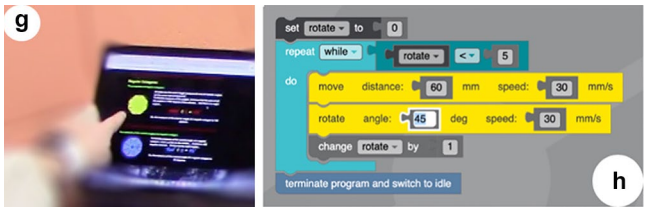
(continued)

TABLE 5 (CONTINUED)

Scenes	The pair converses . . .
	Barbara [01:20]: So you want to try 110? *(She changed the angle value in the rotate block to 110. ⓒ)*
	
	Lillian [01:27] Sure.
Scene 2	Barbara [01:27]: So we just gotta guess numbers, ok. Lillian [01:30]: Or we can cheat and say *(She opened up a Google search page.)* Barbara [01:32]: Oh, how do you code? Lillian [01:33]: What, what degree is an octagon angle? *(She typed her utterance, "what degree is an octagon angle" in her Google search.)* Barbara [01:37]: You are right. *(She loaded the code with the angle value of 110 degrees {which she had guessed}.)* *(Lillian completed her Google search about the angle of an octagon.)* Lillian [02:14]: 135 *(Lillian read the angle information from the webpage she found about octagon angles.)* Barbara [02:16]: Yeah, I'll just load this *(She changed the angle to 135 ⓓ but a popup message appeared stating that the numeric value should range only from −128 to 127; then OzoBlockly automatically replaced 135 to 127)*. Let's do it.
	
	Lillian [02:17]: Sorry. *(Lillian continued reading the webpage including the information of the interior angle (135°) and central angle (45°) values of an octagon)* Barbara [02:17]: Oh, you are good *(Barbara loaded the code with 127° and ran the Ozobot on the octagon shape. The Ozobot followed one side but failed to trace the shape because it veered to the inside of the octagon shape.)*
Scene 3	Lillian [03:05]: Maybe it's not a regular octagon. And also how long do we think this is? *(Lillian questioned about the length of the octagon sides as she pointed with her hand on one of the sides of the shape (the left side of the photo). Barbara (the right side of the photo) counted the sides of the octagon shape. ⓔ)*
	
	Barbara [04:06]: *(She attempted to replace the angle value, 127, with 135, but OzoBlockly automatically changed back to 127 as in Scene 2. She only changed the distance value from 50 to 70. ⓕ)* I'm trying a distance of 70. Lillian [04:09]: Oh, ok.
	

(continued)

TABLE 5 (CONTINUED)

Scenes	The pair converses . . .
	Barbara [04:49]: Stupid. *(Loading the code to the Ozobot did not work momentarily)* There we go. Lillian [04:59]: Yay. *(Barbara loaded the code with 70 mm for the distance and ran the Ozobot on the octagon shape. The Ozobot followed one side incorrectly because the distance was not correct and failed to trace the shape because it veered to the inside of the octagon shape.)* Barbara [05:19]: That's not it.
Scene 4	Lillian [05:19]: It's a little too much. So maybe 60. *(She referred to the distance value.)* Barbara [05:25]: 60 is distance, but the angle . . . Lillian [05:27]: The angle is crazy, but oh wait *(Lillian was on the scaffolding, then she moved to the octagon information webpage and saw the central angle value part of the webpage while trying to open OzoBlockly to see the code)*, the internal angle is 135 then what's the [exterior angle]? You know what I mean? Would it be 180 minus 135? *{Based on her reading of the webpage about octagon angles, she questioned if the angle value in the rotate block should be for an exterior angle.}* Barbara [05:54]: We can try it. Lillian [05:57]: Because it's like this is 135 the inside *(Lillian showed the interior angle of an octagon on the webpage* ⓖ*, and Barbara changed the angle to 45.* ⓗ). But if we're doing it on the outside, I don't know, if it's different you know.  Barbara [06:07]: Cool, cool, cool, cool. Lillian [06:08]: I have no idea, but. Barbara [06:10]: We'll try it. I mean it's a good idea. A [as] good [an] idea as any *(Barbara started loading the code.)*. I don't know what to do. *(Barbara ran the Ozobot on the octagon shape.)* Lillian [06:38]: Oh oh. *(Barbara and Lillian looked at the Ozobot while it was moving.)* Barbara [06:41]: Ahhh! *{Barbara expressed her excitement because the Ozobot did not veer off to the inside of the octagon shape.}* Lillian [06:42]: We got it. Lillian [06:48]: Wow. That was good. *(Barbara and Lillian continued to look at the Ozobot while it was tracing the octagon shape and high-fived.)* *{Their change to 45° fixed the bug in the rotate block, but the distance and the number of rotations were incorrect in the code.}*

Note. Digits within brackets are time stamps. Text within brackets represents (a) missing word(s). Text in parentheses is the researchers' description of the participant's actions or the robot's movement that occurred at the same time as the transcribed speech in the immediately preceding sentence. Text within curly brackets is the researchers' description of the function of the dialogue and/or action.

Harper once had. But Barbara and Lillian applied the rule of following the lines making an obtuse angle not only by observing but also by learning/applying a general rule of octagon angles. Lillian researched octagon angles on the internet, and then the pair tried 135° (Scene 2). OzoBlockly automatically replaced 135° with 127° due to the limited range of angle values. When changing to 135° failed to resolve the bug, their explanation was that the travel distance, 50 mm, was insufficient (Scene 3). While unsound, the explained cause appeared reasonable to the pair. This probable explanation was discarded after noticing that the robot still did not complete the octagon even after increasing the travel distance (70 mm).

While trying to make sense of why the increased distance did not fix the bug, Barbara and Lillian noticed that the robot veered off the octagon, especially the angle (Scene 4). Then they decided to try the exterior angle rule (Scene 4). Their attempt at *sensemaking while doing* with the increased distance as well as 135° led them to change the angle value to 45° (= 180° − 135°) in the rotate block. This sensemaking process seems to have enabled only one change in the angle from 135° to 45°, which contrasts with Charlotte and Harper's debugging that involved multiple changes from 110° to 50° and again to 45°. Amelia and Elena also went through multiple changes after their use of an obtuse angle. That is, they changed the angle from 70° to 120°, then to 20°, then to 25°, and again to 40°. As shown in Table 6, there was no conversation to make sense of the unexpected result. They quickly concluded that the angle value should not be bigger but less than 70°, without sensemaking after their change to 120° other than depicting the result as follows: "we made it go too far now".

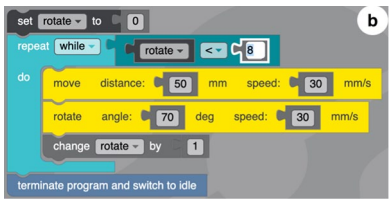
TABLE 6
Amelia and Elena's Octagon Debugging Episode

Scenes	The pair converses . . .

Scene 1 *(Elena loaded the code and ran the Ozobot on the octagon shape. Amelia and Elena observed the Ozobot's movement.)*

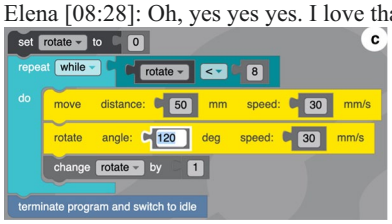
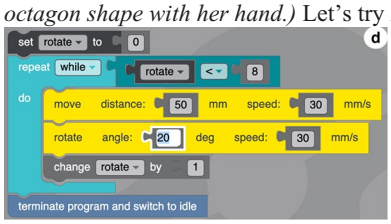
Amelia [05:34]: Ok, goes once, twice . . . (*She counted the Ozobot's turns.)*
Elena [05:37]: Oh, wait. It's making one, but it's not on the path.
Amelia [05:43]: I feel like it's, I don't think it was eight. And also, I think it wasn't like wide enough.
(Elena ran the Ozobot on the octagon shape again and Amelia and Elena reexamined its movement.)
Elena [05:57]: One.
Amelia [05:57]: Two, three, four, it was five. Six?
Elena [06:07]: It was five. It just rotated back around. *{She stated this because the Ozobot did not trace the octagon shape accurately.}*
Amelia [06:07]: Alright.
Elena [06:09]: So then, do we have to, you could do this. *(She clicked the number of rotate counts in the code.)*
Amelia [06:14]: I would say we would move that [the number of rotations, 5] to 8, but then we also have to do the angle. *(She pointed at the angle part in the code.* ⓐ*)* So, what I just want to say *(while typing in the scaffolding as a response to its prompt asking, "Your Hypothesis 1?")*, change the repeat while rotating to . . . What was it? 8? *(She asked about the number of rotations.)*

Elena [06:38]: Yeah, sorry. *{She apologized for her belated response to Amelia's question about the revised number of rotations, 8.}*
Amelia [06:40]: Repeat [while rotate] less than? *(Elena looked at the scaffolding prompts and responses on Amelia's screen.)*
Elena [06:42]: Yeah, yeah. Um, repeat one, rotate. Wait. Rotate. I don't know. I think we keep that there because it was rotating five times. *{She said this to explain the reason why they needed to change the number of rotations from 5 to 8, which was because the Ozobot was rotating five times.}*
Amelia [06:55]: Yeah.
Elena [06:55]: And then move distance. *(She pointed at the move distance block with a cursor.)*
Amelia [06:58]: Wait. So, we're doing this because the Ozobot does not complete 8 lines [of the octagon]? *{She asked the reason for the change in the number of rotations from 5 to 8 in order to respond to the scaffolding prompt.}*
Elena [07:03]: Correct. *(Amelia started typing "The Ozobot does not complete 8 lines" as their response to the scaffolding prompt asking, "Reason for Hypothesis 1?")* I think we can do that first and then figure out what else we need to do in a sense like that can be our first hypothesis. *{She indicated that changing the number of rotations in the repeat while block from 5 to 8 could be their first hypothesis.}*
Amelia [07:13]: For the second one [hypothesis], we have to change, yeah yeah.
Elena [07:16]: Yeah.
Amelia [07:16]: Let's change ours first. *(She suggested changing the code first and testing the code before responding to the scaffolding prompt asking, "Your Hypothesis 2?")* Change it to, wait, do we change it to eight? *(She pointed at the number of rotations in the code.) {She stated this because she noticed that Elena has not changed their code yet according to their discussion.}*
Elena [07:20]: Oh, we do not. *(She changed the number of rotations from default, which was 5, to 8.* ⓑ*)*



(continued)

TABLE 6 (CONTINUED)

Scenes	The pair converses . . .
	Amelia [07:23]: If this does not work, I'll just give up because [inaudible]. *(Elena loaded the code with 8 rotations and the Ozobot made certain sounds while Elena was loading the code.)* Elena and Amelia [07:33–7:39] *{Researchers removed unrelated conversations about sound they heard from other Ozobots in class.}*
Scene 2	*(Elena ran the Ozobot on the octagon shape. Amelia and Elena observed the Ozobot's movement.)* Amelia [07:41]: One, two, three, four, five. Elena [07:54]: Six, seven. You are right because when it gets here it goes in a 90-degree angle. *(She pointed at a vertex of the octagon shape.)* Amelia [08:03]: Yeah. So, what is that? 120? Elena [08:05]: What? I think it was 90-degree angle, it goes like that. *(She recreated the pathway the Ozobot traveled on the octagon shape with her hand.)* Amelia [08:09]: Yeah, so are these 120 degrees? *(She pointed at a vertex of the octagon shape.)* Elena [08:17]: You're probably right 'cause this is 90, this is 180. *(She visualized the angle on the paper with her finger.)* It has to be 120. So then where do you put that [120 degrees] [in the code]? Amelia [08:24]: In the angle? *(She pointed at the angle part of the code on the computer screen.)* Elena [08:28]: Oh, yes yes yes. I love that idea. *(She changed the angle from 70 to 120 degrees in the code. ⓒ)*  Amelia [08:31]: Rotate was at 70 degrees and we are changing it to 120 degrees because it was turning too sharp of [inaudible] *(She typed her utterance as a response to the scaffolding prompts asking "Your Hypothesis 2?" and "Reason for Hypothesis 2?")* Amelia [09:10]: Octagon. Alright. *(She finished typing their response to scaffolding prompts.)* Elena [09:18]: Let's do it. *{She expressed the pair's readiness to run the revised code.}* Amelia [09:20]: Hopefully, it works. Elena and Amelia [09:22-10:09]: *{Researchers removed off-topic conversations.}*
Scene 3	*(Elena loaded the code with 120 degrees in the angle value and ran the Ozobot on the octagon shape. Amelia and Elena observed the Ozobot's movement.)* Elena [10:10]: What? *{She expected that the angle change would lead to the Ozobot's correct tracing of the octagon shape but the Ozobot turned too sharply and went off the octagon shape.}* Amelia [10:16]: So, maybe we made it go too far now. So maybe it has to be less than 70. Elena [10:20]: Yeah. Yeah. Okay. You can stop right there. Amelia [10:22]: It maybe. Elena [10:25]: Let's try like . . . Amelia [10:26]: What about like 30 or 20? Elena [10:28]: Because we go here and it literally just starts to go like that. *(She recreated the pathway the Ozobot traveled on the octagon shape with her hand.)* Let's try 20. *(She changed the angle from 120 to 20 degrees in the code. ⓓ)*  Amelia [10:33]: 20. Okay. I'm not going to write it down yet 'cause I can always change that one. *{She stated that she would not write down the angle value change to 20 degrees as a hypothesis in their response to the scaffold until they know the change worked or other change is needed.}* Elena [10:37]: Do we have to have three? *{She asked about the number of hypotheses they were asked to generate in the scaffold.}* Amelia [10:37]: Yeah. Who knows if this will work? *{She said this to reiterate the need to see if the angle value, 20, works.}*

(continued)

TABLE 6 (CONTINUED)

Scenes	The pair converses . . .

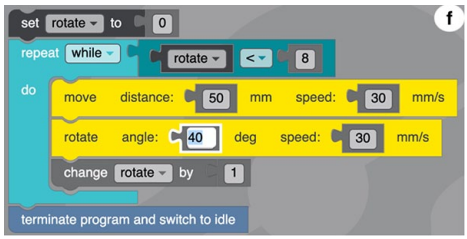
Scene 4 *(Elena loaded the code with 20 degrees in the angle and tried to run the Ozobot on the octagon shape.)*

Elena [11:01]: Oh, I just turned it off.
Amelia [11:02]: Oh, I was going to say, what did we do wrong?
Elena [11:04]: No. *(She turned the Ozobot back on and ran it on the octagon shape again.)*
(Elena and Amelia observed the Ozobot's movement.)
Amelia [11:06]: Oh, no. Why is it like that? *{She said this because the Ozobot started to make sounds and blink with colorful lights, instead of making any movement.}*
Elena [11:07]: I don't know what I do to make it [the Ozobot] freak out every time.
Amelia [11:11]: It's only yours that freaks out, though, I swear.
Elena [11:17]: I don't think it did it. I think that it just didn't calibrate again. *(She pointed at the calibration platform in the OzoBlockly program.* ⓔ*)* Let's try calibration 'cause that's . . .

Amelia [11:21]: That's the same exact thing. *{She explained that the Ozobot's movement was exactly the same with the movement before they changed the angle to 20 degrees.}*
Elena [11:23]: That's more than just . . .
Amelia [11:24]: There's no way that's 20 degrees. *{She reiterated that the angle of the ozobot's rotation was not 20°.}*
Elena [11:27]: Yeah, I agree. Let's re-calibrate it. *(She calibrated the Ozobot again.)* Oh there it goes. Alright, perfect. *(The calibration was successful and then she loaded the code.)*
(Elena ran the Ozobot on the octagon shape, and Elena and Amelia observed the Ozobot's movement.)
Amelia [12:00]: What the heck? *(With expressions of shock, Amelia and Elena observed the robot spinning its wheels immediately after being taken off of the calibration circle.)*
Elena [12:05]: *(She placed the Ozobot on the octagon shape and attempted to start the program.)*
Elena [12:15]: No, stop. *(She told the Ozobot to stop.)*
Amelia [12:18]: Maybe it didn't calibrate 'cause it's like turning back on.
Elena [12:21]: You know, you're right. Oh, okay.
Amelia [12:22]: Oh my gosh.
Elena and Amelia [12:24–14:59] *{Researchers removed off-topic, unrelated personal conversations.}*
Amelia [15:00]: Ok. Maybe let's try calibrating it again.
(Elena began to calibrate the Ozobot again.)
Elena [15:20]: Ok. No! Sorry, got really upset at me for a sec. *{She stated that the Ozobot was upset because she made a mistake of not clicking the power button on the Ozobot when calibrating it.}*
Amelia [15:38]: Are we gonna have to troubleshoot? Why would we ever need to do this?
Elena [15:45]: It's just part of coding [inaudible].
(Elena tried to calibrate the Ozobot again.)
Elena [16:04]: Is it [Ozobot] growling at me? *{She said this because of the sound that the Ozobot made during calibration.}*
Amelia [16:05]: Yeah, it's like we don't want to play with this because it's too hard.
(Elena tried to re-calibrate the Ozobot three more times.)
Elena [16:55]: Oh.
Amelia [16:55]: Oh.
Elena [16:56]: Oh, my God. [inaudible] *{She said this because she was finally able to complete the calibration after multiple trials.} (Elena loaded the code.)*

(continued)

TABLE 6 (CONTINUED)

Scenes	The pair converses . . .
Scene 5	*(Elena loaded the code with a 20 degree angle and ran the Ozobot on the octagon shape. Amelia looked at the Ozobot's movement.)* Amelia [17:17]: Oh, gosh. It doesn't do it enough. *{She explained that the Ozobot's rotation was not big enough to follow the octagon shape on the map.}* I'll try this. *(She took over Elena's computer and started working on the code.)* Elena [17:21]: *(Elena stepped out for a while.)* Amelia [17:25–19:32]: *(Amelia changed the angle in the code to 25 degrees and reloaded the code, and ran the Ozobot by herself. She then changed the angle to 40 degrees, ran the code again, and then typed their response to the scaffolding prompts asking their second hypothesis and the reason for the hypothesis.)* Elena [19:31]: *(Elena came back)* So, do you, did it do anything? Amelia [19:33]: I think I got it to work. *{She said this to report that she fixed the problem with the angle value, 20, by changing it to 40 degrees. ⓕ}* So it's not perfect, but look, it's pretty good. *(Amelia ran the Ozobot with 40 degrees angle and showed Elena the Ozobot's movement.)* Elena [19:43]: Okay. How did you do that? Amelia [19:47]: So, it's not like 100%, but it works pretty well. So, I feel like . . . Elena [19:54]: I think it's good. Amelia [19:55]: That's fine. Elena [19:56]: No, I feel like, yeah. Amelia [19:59]: Yeah. Okay. *(She returned to working on the scaffolding prompts.)* Elena [20:20]: So, 40 degrees worked better *(She looked at the code on the screen that Amelia had changed)* than 20, 'cause 20 it was too little. *(She visualized the angle on the paper with her finger.)* 

Note. Digits within brackets are time stamps. Text within brackets represents (a) missing word(s). Text in parentheses is the researchers' description of the participant's actions or the robot's movement that occurred at the same time as the transcribed speech in the immediately preceding sentence. Text within curly brackets is the researchers' description of the function of the dialogue and/or action.

Theme 3: Reflective Abstraction Was Seldom Observed

Scaffolding Reflective Debugging But Not Reflective Abstraction. When the code was debugged, participants seldom attempted to generalize explanations for why some code revisions worked and some did not. For example, when changing the angle value to 110° did not fix the code, Charlotte and Harper explained why they changed the angle in their response in the debugging scaffold: "Because the 110 degrees was making our Ozobot take even sharper turns than the 70-degree angle it was taking before, we thought we should make the angle less than 70." However, while tinkering from 70°, to 110°, 50°, and 45°, a collection of specific explanations for multiple changes in the angle value did not lead to a generalizable explanation for why bigger angles made sharper turns or why 45° worked but 50° did not. As shown in Table 3, Scene 5, when debugging was completed, the pair was inquisitive about why using 8 for number of rotations did not work until the turn angle was fixed but their sensemaking attempt did not reach an explanation that could account for relationships among multiple cases and results. This finding does not seem unnatural considering that mostly tacit rules were used (see Theme 1).

The current participants did not have enough experience programming and debugging to engage in inductive reasoning. Still, our findings prompted us to reexamine the scaffold design intended for reflective debugging. In the scaffold, participants were invited to describe and reflect on what *actually* made their robot behave unexpectedly and in what ways the code was fixed when debugged. The design intent was to scaffold them to reflect on the bug location and resolution and solidify their understanding of how the code worked (Kim et al., 2018). Most responses accurately summarized the bug location and resolution. For example, Charlotte and Harper's responses are as follows:

> Prompt: What was the real problem that made your Ozobot behave unexpectedly?
> Response: The angle and the amount of turns it was making was off.

Prompt: How was the problem fixed?
Response: We changed the angle to 45 and changed the repeat while rotate < 8.

In the scaffold, participants were also asked *why* certain changes were made. For example, Charlotte and Harper reflected on their changes in the repeat while block: "We tried it [8] before we changed the angles and it did not work, so we changed it [8] back to 5, and then back to 8 after we changed the angles . . . [because] it was only turning 5 times." The *why* questions seem to have invited participants to describe cases and results that they were seeing without explaining relationships among multiple cases and results. For example, participants were asked to describe each case *N* and result *N* and their relationship, but not the relationship *among* cases and results:

Case 1 (repeat while rotate < 5 & angle 70 degrees) → Result 1 (the robot did not work properly)
Case 2 (repeat while rotate < 8 & angle 70 degrees) → Result 2 (the robot did not work properly)
Case 3 (repeat while rotate < 8 & angle 45 degrees) → Result 3 (the robot worked properly)

Our scaffolding is not the only tool to invite students learning computer science to ask why questions; for example, another such tool is Whyline, which invites students to ask why questions and points to areas of the code that could provide answers (Ko & Myers, 2008). In the current study, it appears that the scaffolding served as a sociocultural mediator (Belland & Drake, 2013) of the pairs' work, thereby fostering abductive reasoning and, by extension, cautious tinkering. Scaffolding for participants to review the relationship *between* case *N* and result *N* seems to have played a facilitative role in cautious tinkering of participants as hinted by Charlotte in the interview:

> What changes you made and then what happened after the changes were made, writing that down helped. I think completely writing down the code was a lot [of help] . . . I think because [while] writing down what you changed and then writing down what the change did, you could see that [the change] did the opposite of what we were supposed to do. So let's try the other way.

Similarly, Amelia considered such scaffolding helpful as noted in her interview: "I think having the hypothesis and then writing the reason why kind of helped in debugging it 'cause you were thinking more of how do I fix it when it's on there." In the interview, Elena also explained the role of hypotheses:

> I think the hypotheses part to write it down [was the most helpful], to really see what my process of thinking was. It was good to understand where my line of thinking was starting to go and it was good to have a second hypothesis and a third if that didn't work out either.

Data suggest that reflective debugging was scaffolded but reflective abstraction was not. Reflective abstraction is the process of arriving at new knowledge through reflection on current and earlier actions, operations, construction, and reconstruction (Abrahamson, 2012; Cetin & Dubinsky, 2017; Piaget & Inhelder, 1969). Reflection on multiple, isolated relationships between each case *N* and result *N* did not lead to general rules that may explain the relationships across cases and results. As figural and numerical cues were used in the pathway from abduction to induction about algebraic patterns among ECE majors (Rivera & Becker, 2007), the scaffold of the present study could be redesigned to facilitate the use of multimodal cues in more holistic ways. It may be worthwhile to use a list of cases and results to not only summarize the relationship per case but also visualize multiple relationships as a whole as in inductive debugging (Myers et al., 2004). The following rules, for example, could be reasoned when changes in the code (cases) and robot behaviors on the change (results) are reviewed altogether and reflected on:

Rule X: The rotate angle value should be equal to the exterior angle of a regular octagon.
$\therefore 360°/8 = 45$
Rule Y: The number of rotations should be the number of sides of a regular octagon.
$\therefore 8$
Rule Z: To rotate 8 times, the numeric value in the "repeat while *rotate* <" segment should be 8 because the initial value for the *rotate* variable is 0.
$\therefore$ repeat while rotate < 8

Actual Use of the Scaffold Limited Sensemaking. Participants often used the scaffold to report what they had done. For example, Barbara and Lillian completed the octagon debugging task and then answered scaffold prompts that were designed to be used during the debugging process. Charlotte and Harper listed only some changes they made in the buggy code; they listed 70° to 110°, and then to 45°, instead of listing all the angle changes from 70°, to 110°, 50°, and 45°. Barbara and Lillian made an error in their final specification of the bug location and resolution by including the change in the robot travel distance that they ended up excluding in their final code. Amelia and Elena responded to the scaffold as they engaged in debugging more often than others, as exemplified in Scene 1 of Table 6, but they only recorded the last change they made (i.e., 40°) rather than listing all changes made in Scenes 2 through

5. Although seemingly trivial, these cases may have also discouraged reflective abstraction. While the scaffold helped participants think back on what they had done, using the scaffold as a retrospective tool may have partially resulted in little sensemaking *while* doing (see Theme 2 above). The scaffold was designed in such a way that all changes needed to be reflected on, not just those that led to bug resolution. Accumulated reflections could lead to reflective abstraction (Abrahamson, 2012). If inductive reasoning can be attempted after each conclusion of abductive reasoning, the transition from abduction to induction (Rivera & Becker, 2007) could be possible sooner than later. This means that participants would have more repertories of potential rules that can be used in future abductive reasoning. Considering that participants applied their octagon debugging experience when asked to explain why the robot was not tracing a hexagon during the interview, inductive reasoning about the repeat while rotate $< n$ sides block may have been initiated when they saw familiar code. The interview with Harper, for example, shows that she mimicked octagon debugging for the hexagon code. She talked about her debugging approach without much exploration.

> Harper: I remember doing something like this. I remember we at first tried to first change this [the number in the repeat while block] to . . . well, hexagon is six sides, right? Yeah. So, we changed that to eight [in the octagon debugging]. But it still wouldn't work because of the angles were off. So I think I would just change one of these, I guess this angle [in the hexagon code]. I don't know what I would change it [the angle value] to. I would do trial and error and see, and then change the sides to six to see if it could make a hexagon.
>
> Interviewer: Why do you think this angle could be a problem?
>
> Harper: When we did it before [in the octagon debugging] the angle was too sharp and so before [Ozobot] can even get to eight sides, it already closed itself off.
>
> Interviewer: Do you think if you change to six and then run it, do you think the angle should be smaller or bigger than the number that is there?
>
> Harper: Smaller? Hmm. I think.
>
> Interviewer: Why do you think so?
>
> Harper: Because when I did this [in the octagon debugging], I remember we changed it to like 110 or something like that cause we thought it would be an obtuse angle but it did the opposite. It [made] even sharper turns.

Turkle and Papert (1990) argued that tinkerers may become decisive when they work on familiar programming tasks, which they regarded as "the benefits of their long explorations" (p. 141). However, Harper's sensemaking of the repeat while block seems isolated from sensemaking related to the angle and distance. Reasoning that the angle should be less than 70° in the hexagon task just because 70° in the octagon task had to be reduced is evidence of lack of reflective abstraction. There was a bug in the distance, but Harper did not mention making any change in the distance probably because no change was needed in the distance in the octagon task. If sensemaking was completed while and after debugging (see Theme 2), rules about the angle may have been applied to justify the smaller angle value in fixing the hexagon code.

Theme 4: Generating Multiple Hypotheses in Advance Was an Unnatural Requirement

Scaffold Design Intent Misaligned With Abductive Reasoning. The scaffold design was grounded in the literature highlighting the critical role of hypothesis formulation in debugging (Araki et al., 1991; Kim et al., 2018). Along with prompts for generating hypotheses, the scaffold guided participants' attention to specific structures of the code, blocks, and numeric or descriptive value in blocks. For example, the following questions were given during hypothesis generation: Is the sequence of blocks arranged correctly? Is the numeric value in blocks aligned with what you want your Ozobot to do? Is the logic block used correctly? Scaffolding design suggested in Kim et al. (2018) included use of common errors as part of scaffolding for debugging block-based code, but not as a checklist for step-by-step execution. Thus, these questions were intended to draw participants' attention to common errors. Lillian noted that the questions prompted her and her partner, Barbara, to pay attention to the code in a productive way:

> The questions that were asked [were helpful] *(The interviewee is looking at the printed copy of her responses to the scaffold)*. I think the numeric value in the blocks was most helpful. . . . So when we looked at it, that's the one that sort of tipped us off to address the issue. The one question is, is the descriptive value in blocks aligned with what you want your Ozobot to do? And then the numeric values . . . our first hypothesis was that the distance was wrong. I don't think that ended up being right. I'm not exactly sure, but the issue was the angles. So when we started looking at the degrees and the numeric values, with that question, it helped us pay attention to something that we may have missed otherwise.

Regardless of the design intent for hypothesis generation, the scaffold seems to have asked for an unnatural course of activities in that two or three hypotheses had to be written down altogether before testing any. Considering the salient use of abductive reasoning among our participants, this part of generating multiple hypotheses in advance seemed misaligned with their reasoning process. During reasoning through abduction, a new hypothesis is realized when there is new information that demotes the probability of the old hypothesis (Magnani, 2009). That is, it is natural that the second probable cause (Hypothesis 2) explaining the robot's malfunctioning could not be guessed without testing the first probable cause (Hypothesis 1). Thus, the requirement of generating multiple hypotheses in advance may have been misaligned with the participants' reasoning approaches:

> Interviewer: Which parts of the scaffold most conflicted with your thinking processes or were not aligned with your thinking processes?
> Lillian: I feel like it was hard to list out specific hypotheses 'cause we sort of did a bunch of trial and error. So to specifically look at what the problem was there. It could have been a lot of things, which I guess is why it said reason for multiple hypotheses . . . I also think that we are anxious to just start moving.

Charlotte noted that her partner, Harper, and she responded to the scaffold retrospectively because they wanted to see what a change would do first before listing multiple hypotheses:

> Charlotte: I guess that hypothesis [was most conflicted] 'cause . . . I think I'm a more visual person. . . . I'm better at just like trial and error . . . I would have just like tried a bunch of things and see what worked instead of writing something out.
> Interviewer: How did you handle with that process of writing hypotheses?
> Charlotte: 'Cause my partner I think was kind of the same way, we actually would kind of change something and then . . . write a hypothesis for that. We would kind of do it a little bit backwards actually. We would change something and then I'd be like, okay, we have to write a hypothesis for what we changed and then do it.

Amelia and Elena exhibited more concurrent engagement with the scaffold and debugging than other pairs, as discussed earlier (Theme 3). Still, they did not write multiple hypotheses in advance. They wrote one and tested it prior to writing another one, as shown in Table 6. Moreover, when their change to 120° did not work, they decided not to write another hypothesis yet. That is, they chose to see what happened with their new change (20°) before writing their third hypothesis related to the change. Their first hypothesis was about the number of rotations (Scene 1). Their second hypothesis was about the angle change from 70° to 120° (Scene 2). Amelia explained to Elena that she was not writing down their third hypothesis related to the new angle value, 20°, because they may need to abandon the hypothesis of the moment if 20° does not fix the bug (Scene 3).

Natural Process of Abductive Thinking Leading to Tinkering. Charlotte defined her problem-solving during debugging as a process of trial and error:

> I think it [debugging] is very similar [to other problem-solving] 'cause a lot of things are trial and error. I don't fix much like cars, fridge, refrigerators. But even the fridge . . . in my apartment, there's like 7 and 1, but I didn't know which one was cold[est] and which one was the warmest. So you just try 7 and wait a few hours and see if it gets colder or warmer. So it's just like trial and error.

Positive notions were often expressed about trial and error used in debugging and problem-solving in general. Harper wrote in her reflection:

> We thought about what we thought the solution would be, tried it out, failed, and kept trying until we solved it. This is how a lot of things in life work, and many math problems as well. If something is wrong with my phone, I think "maybe if I reset it, it will work," then if it doesn't, I try other things, and if that doesn't work, I keep trying things until I find the solution that fixes my phone.

She also wrote that she would apply her debugging experience to teaching young children as follows: "I would teach them how failing is not a bad thing, it takes many tries to solve something and that is okay." Participants *chose* to make a series of incremental changes based on tangible observations. Lillian seemed to be more of a planner usually, but for debugging, she tinkered, as she noted here:

> The approach we took was more trial and error so see if it worked and not. And, in my real life I try to plan things out more specifically, but to visualize it, it was easier to make the changes as we were going.

These findings are similar to the study of Ko et al. (2019) in which most participants still chose to tinker even though their intervention was designed to help novice programming learners debug step-by-step in structured ways. Now knowing our participants' common practice of abductive reasoning, the design feature that required them to come up with multiple hypotheses with no explorations yet seems far from our intentions.

As mentioned earlier, tinkering in the present study was cautious (Berland et al., 2013; Perkins et al., 1986) and intentional (Fitzgerald et al., 2010). For example, as Amelia and Elena explained in their interviews, their incremental changes were grounded in tangible observations of the robot's behaviors:

> Usually, I would look through the code and see what I didn't want to do or what I had planned it to do and then based off of what it actually did, I would add something or try to reprogram it to make it better. (Amelia)

> When I would program it one way for example, like turning and going straight along the line, and it wouldn't work the way I expected it to, it wouldn't turn enough or whatever. I would have to readjust that . . . We went over and double-checked everything and kind of correlated where the code was with the movement that we saw it doing. So we would play it again, play the code again, let the Ozobot move again. We would check on the code, everything that it was doing and want to stop doing what we wanted it to. We would figure out where we were in the code and use that. (Elena)

The function of the hypothesis-driven scaffolding was not misaligned with our participants' dominant reasoning (abductive reasoning) possibly because they ended up formulating multiple hypotheses while tinkering, instead of formulating them all at once. Their hypotheses (Table 7) had explanatory value. Abduction is a "process of reasoning in which explanatory hypotheses are formed and evaluated" (Magnani, 2009, p. 8). Considering that abductive reasoning

TABLE 7
Responses to Multiple Hypothesis Generation Prompts in the Debugging Scaffold

Prompts	Charlotte and Harper	Barbara and Lillian	Amelia and Elena
Your Hypothesis 1	We need to change the rotate < 5 to rotate < 8.	Distance was off.	[We need to] change the repeat while rotating to 8.
Reason for Hypothesis 1	The Ozobot was only moving 5 sides and an octagon has 8 sides.	It didn't go far enough before turning.	The Ozobot does not complete 8 lines.
Your Hypothesis 2	We need to [change] the angle of our Ozobot, so we changed it to 110 degrees.	Angle was off.	The rotate was at 70 degrees and we are changing it to 40 degrees.
Reason for Hypothesis 2	The Ozobot was originally making too sharp of turns, so we thought we needed to change it from 70 degrees to an obtuse angle, which would be 110.	We knew it looked bigger than 90 degrees on the inside.	The Ozobot was turning at too sharp of an angle for it to be an octagon.
Your Hypothesis 3	We change the angle from 110 degrees to 45 degrees.	Ozobot stopped before completing the shape.	
Reason for Hypothesis 3	110 was way too big of an angle so we realized we actually had to choose a smaller angle than 70 degrees, not a larger angle.	We extended the distance to 70.	

"never reaches the status of best hypothesis" and that hypotheses can be partial and elementary as long as there is some plausibility (Magnani, 2009, p. 11), all these hypotheses are reasonable explanatory hypotheses. As shown in Table 7, Barbara and Lillian reported their observation of the robot's current movement as hypotheses. Charlotte and Harper as well as Amelia and Elena wrote their plan for change in the code as hypotheses. Considering that "explanation deals with the current or past problems, while prediction deals with future problems" (Abe, 2003, p. 232), Amelia and Elena may have attempted to generate a predictive hypothesis even though the scaffold asked for hypotheses for why the robot *was behaving* incorrectly. Hypotheses used in deduction are for prediction (Magnani, 2009). Most hypotheses were still of explanatory value in that they were used in searching for best explanations during their tinkering. Nonetheless, since ECE teacher candidates are usually exposed to deduction and induction, not abduction (Rivera & Becker, 2007), the scaffold could be redesigned to guide participants to distinguish explanatory hypotheses from predictive hypotheses and maximize the benefits of their use of explanatory hypotheses.

Conclusion and Implications for Computing Education

The unique contribution of this article is the finding of abductive reasoning used in tinkering. Unlike many existing studies that attempt to identify deficits that lead to tinkering, we investigated how novice programmers tinkered so that we could understand their *reasoning* processes. We made constant efforts throughout this research to avoid assigning value to tinkering or attributing the use of tinkering to participants' low or nonexistent prior programming knowledge and experience. This may sound counterintuitive considering this article indeed reports the tinkering process of novices, but the lens of ethnomethodology enabled our investigation of tinkering as in and of itself without value judgment.

Another implication is that the study findings point to a way to include further instruction on reasoning in ECE contexts. A common criticism is that little reasoning is learned in early learning curricula, even in mathematics education in which reasoning is essential (Stylianides et al., 2013). This criticism is applied not only to ECE but also to preservice, ECE teacher education. For example, inductive and deductive reasoning is covered in the ECE teacher mathematics education curriculum but often not abductive reasoning (Rivera & Becker, 2007). But the relationship between abduction and induction is critical to generalization (Rivera & Becker, 2007). If teachers do not recognize abductive thinking, the potential process to induction and generalization could be discarded (Abrahamson, 2012).

Specifically related to scaffolding design for computing education, this study suggests that hypothesis-driven scaffolds can be instrumental to reflective debugging, but it is important to consider the reasoning processes of computing learners. After reviewing data from the present study, we redesigned our scaffold to support reasoning by abduction and prompt for reflective generation of rules during abductive reasoning processes. This redesign is meant to scaffold our teacher candidates to see relationships not only *between* individual case *N* and result *N* while reasoning by abduction but also *across* multiple cases and results throughout reason-

ing processes. This redesign decision calls for further asset-based scaffolding research.

Acknowledgments

This research is supported by Grants 1927595 and 1906059 from the National Science Foundation (United States of America). Any opinions, findings, or conclusions are those of the authors and do not necessarily represent official positions of the National Science Foundation.

ORCID iDs

ChanMin Kim https://orcid.org/0000-0001-9383-8846
Brian R. Belland https://orcid.org/0000-0002-8925-9152
Afaf Baabdullah https://orcid.org/0000-0002-2880-0822
Anna Y. Zhang https://orcid.org/0000-0003-0609-4741

References

Abe, A. (2003). Abduction and analogy in chance discovery. In Y. Ohsawa, & P. McBurney (Eds.), *Chance discovery* (pp. 231–248). Springer. https://doi.org/10.1007/978-3-662-06230-2_16

Abrahamson, D. (2012). Rethinking intensive quantities via guided mediated abduction. *Journal of the Learning Sciences*, *21*(4), 626–649. https://doi.org/10.1080/10508406.2011.633838

Akcaoglu, M. (2014). Learning problem-solving through making games at the game design and learning summer program. *Educational Technology Research and Development*, *62*(5), 583–600. https://doi.org/10.1007/s11423-014-9347-4

Aliseda, A. (2006). *Abductive reasoning: Logical investigations into discovery and explanation*. Springer.

Araki, K., Furukawa, Z., & Cheng, J. (1991). A general framework for debugging. *IEEE Software*, *8*(3), 14–20. https://doi.org/10.1109/52.88939

Beckwith, L., Kissinger, C., Burnett, M., Wiedenbeck, S., Lawrance, J., Blackwell, A., & Cook, C. (2006). Tinkering and gender in end-user programmers' debugging. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 231–240). Association for Computing Machinery. https://doi.org/10.1145/1124772.1124808

Belland, B. R., & Drake, J. (2013). Toward a framework on how affordances and motives can drive different uses of computer-based scaffolds: Theory, evidence, and design implications. *Educational Technology Research and Development*, *61*, 903–925. https://doi.org/10.1007/s11423-013-9313-6

Belland, B. R., Gu, J., Kim, N. J., & Turner, D. J. (2016). An ethnomethodological perspective on how middle school students addressed a water quality problem. *Educational Technology Research and Development*, *64*(6), 1135–1161. https://doi.org/10.1007/s11423-016-9451-8

Berland, M., Martin, T., Benton, T., Smith, C. P., & Davis, D. (2013). Using learning analytics to understand the learning pathways of novice programmers. *Journal of the Learning Sciences*, *22*(4), 564–599. https://doi.org/10.1080/10508406.2013.836655

Bers, M. U. (2018). *Coding as a playground: Programming and computational thinking in the early childhood classroom*. Routledge. https://doi.org/10.4324/9781315398945

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education, 72*, 145–157. https://doi.org/10.1016/j.compedu.2013.10.020

Bers, M. U., Seddighin, S., & Sullivan, A. (2013). Ready for robotics: Bringing together the T and E of STEM in early childhood teacher education. *Journal of Technology and Teacher Education*, *21*(3), 355–377.

Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., & Koller, D. (2014). Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, *23*(4), 561–599. https://doi.org/10.1080/10508406.2014.954750

Brennan, K., & Resnick, M. (2012, April 13–17). *Using artifact-based interviews to study the development of computational thinking in interactive media design* [Paper presentation]. American Educational Research Association Annual Meeting, Vancouver, British Columbia, Canada. https://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Cetin, I., & Dubinsky, E. (2017). Reflective abstraction in computational thinking. *Journal of Mathematical Behavior*, *47*, 70–80. https://doi.org/10.1016/j.jmathb.2017.06.004

Çetin, M., & Demircan, H. Ö. (2020). Empowering technology and engineering for STEM education through programming robots: A systematic literature review. *Early Child Development and Care*, *190*(9), 1323–1335. https://doi.org/10.1080/03004430.2018.1534844

Dershowitz, N., & Lee, Y.-J. (1993). Logical debugging. *Journal of Symbolic Computation*, *15*(5–6), 745–773. https://doi.org/10.1016/S0747-7171(06)80011-8

Fitzgerald, S., McCauley, R., Hanks, B., Murphy, L., Simon, B., & Zander, C. (2010). Debugging from the student perspective. *IEEE Transactions on Education*, *53*(3), 390–396. https://doi.org/10.1109/TE.2009.2025266

Garfinkel, H. (1967). *Studies in ethnomethodology*. Prentice Hall.

Grigoreanu, V., Beckwith, L., Fern, X., Yang, S., Komireddy, C., Narayanan, V., Cook, C., & Burnett, M. (2006). Gender differences in end-user debugging, revisited: What the miners found. In *Visual Languages and Human-Centric Computing (VL/HCC'06)* (pp. 19–26). IEEE. https://doi.org/10.1109/VLHCC.2006.24

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199–237. https://doi.org/10.1080/08993408.2015.1033142

Hanson, N. R. (1958). *Patterns of discovery: An inquiry into the conceptual foundations of science*. Cambridge University Press.

Ingram, J. (2018). Moving forward with ethnomethodological approaches to analysing mathematics classroom interactions. *ZDM*, *50*(6), 1065–1075. https://doi.org/10.1007/s11858-018-0951-3

Kim, C., Belland, B. R., & Gleasman, C. (2020). *Playful coding and playful learning among future early childhood educators*. In *Proceedings of the 2020 Meeting of the International Conference of the Learning Sciences* (*Vol. 4*, pp. 2411–2412). ICLS.

Kim, C., Kim, D., Yuan, J., Hill, R. B., Doshi, P., & Thai, C. N. (2015). Robotics to promote elementary education pre-service teachers' STEM engagement, learning, and teaching.

Computers & Education, *91*, 14–31. https://doi.org/10.1016/j.compedu.2015.08.005

Kim, C., Yuan, J., Gleasman, C., Shin, M., & Hill, R. B. (2017). Preparing pre-service early childhood teachers to teach mathematics with robots. In B. K. Smith, M. Borge, E. Mercier, & K. Y. Lim (Eds.), *Making a difference: Prioritizing equity and access in CSCL, 12th International Conference on Computer Supported Collaborative Learning, Volume 2* (pp. 617–620). International Society of the Learning Sciences. https://repository.isls.org//handle/1/299

Kim, C., Yuan, J., Vasconcelos, L., Shin, M., & Hill, R. B. (2018). Debugging during block-based programming. *Instructional Science*, *46*(5), 767–787. https://doi.org/10.1007/s11251-018-9453-5

Ko, A. J., LaToza, T. D., Hull, S., Ko, E. A., Kwok, W., Quichocho, J., Akkaraju, H., & Pandit, R. (2019). Teaching explicit programming strategies to adolescents. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 469–475). Association for Computing Machinery. https://doi.org/10.1145/3287324.3287371

Ko, A. J., & Myers, B. A. (2008). Debugging reinvented: Asking and answering why and why not questions about program behavior. In *Proceedings of the 30th International Conference on Software Engineering* (pp. 301–310). Association for Computing Machinery. https://doi.org/10.1145/1368088.1368130

Kolko, J. (2009). Abductive thinking and sensemaking: The drivers of design synthesis. *Design Issues*, *26*(1), 15–28. https://doi.org/10.1162/desi.2010.26.1.15

Leake, D. B. (1995). Abduction, experience, and goals: A model of everyday abductive explanation. *Journal of Experimental & Theoretical Artificial Intelligence*, *7*(4), 407–428. https://doi.org/10.1080/09528139508953820

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, *41*, 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Magnani, L. (2009). *Abductive cognition: The epistemological and eco-cognitive dimensions of hypothetical reasoning* (*Vol. 3*). Springer. https://doi.org/10.1007/978-3-642-03631-6

Magnani, L. (2015). The eco-cognitive model of abduction: Ἀπαγωγή now: Naturalizing the logic of abduction. *Journal of Applied Logic*, *13*(3), 285–315. https://doi.org/10.1016/j.jal.2015.04.003

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: A review of the literature from an educational perspective. *Computer Science Education*, *18*(2), 67–92. https://doi.org/10.1080/08993400802114581

Metzger, R. C. (2003). *Debugging by thinking: A multi-disciplinary approach*. Digital Press. https://learning.oreilly.com/library/view/debugging-by-thinking/9781555583071/

Murphy, L., Lewandowski, G., McCauley, R., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: The good, the bad, and the quirky: A qualitative analysis of novices' strategies. *ACM SIGCSE Bulletin*, *40*(1), 163–167. https://doi.org/10.1145/1352322.1352191

Myers, G. J., Sandler, C., Badgett, T., & Thomas, T. M. (2004). *The art of software testing*. John Wiley. https://learning.oreilly.com/library/view/the-art-of/9780471469124/

Park, W. (2017). Magnani's manipulative abduction. In W. Park (Ed.), *Abduction in context: The conjectural dynamics of scientific reasoning* (pp. 41–66). Springer. https://doi.org/10.1007/978-3-319-48956-8_3

Peirce, C. S. (1931-1935). *The collected papers of Charles Sanders Peirce* (C. Hartshorne, & P. Weiss, Eds.; Vols. 1–6). Harvard University Press.

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of learning in novice programmers. *Journal of Educational Computing Research*, *2*(1), 37–55. https://doi.org/10.2190/GUJT-JCBJ-Q6QU-Q9PL

Piaget, J., & Inhelder, B. (1969). *The psychology of the child* (H. Weaver, Trans.). Basic Books. http://catalog.hathitrust.org/api/volumes/oclc/16142.html

Rivera, F. D., & Becker, J. R. (2007). Abduction–induction (generalization) processes of elementary majors on figural patterns in algebra. *Journal of Mathematical Behavior*, *26*(2), 140–155. https://doi.org/10.1016/j.jmathb.2007.05.001

Rose, S. (2016). Bricolage programming and problem solving ability in young children: An exploratory study. In *European Conference on Games Based Learning; Reading* (pp. 914–921). Academic Conferences International Limited. http://search.proquest.com/docview/1859715060/abstract/F8840583576A4E48PQ/1

Searle, K. A., Fields, D. A., Lui, D. A., & Kafai, Y. B. (2014). Diversifying high school students' views about computing with electronic textiles. In *Proceedings of the Tenth Annual Conference on International Computing Education Research* (pp. 75–82). Association for Computing Machinery. https://doi.org/10.1145/2632320.2632352

Stylianides, G. J., Stylianides, A. J., & Shilling-Traina, L. N. (2013). Prospective teachers' challenges in teaching reasoning-and-proving. *International Journal of Science and Mathematics Education*, *11*(6), 1463–1490. https://doi.org/10.1007/s10763-013-9409-9

Turkle, S., & Papert, S. (1990). Epistemological pluralism: Styles and voices within the computer culture. *Signs; Chicago*, *16*(1), 128–157. https://doi.org/10.1086/494648

Turkle, S., & Papert, S. (1992). Epistemological pluralism and the revaluation of the concrete. *Journal of Mathematical Behavior*, *11*(1), 3–33. https://doi.org/10.5642/hmnj.199201.07.08

Zeller, A. (2009). *Why programs fail: A guide to systematic debugging* (2nd ed.). Elsevier. https://doi.org/10.1016/B978-0-12-374515-6.X0000-7

Authors

CHANMIN KIM is associate professor of Learning, Design, and Technology, and Educational Psychology at Pennsylvania State University, University Park. She researches teacher learning of STEM that contributes to the creation of equitable learning environments. Her current focus is on using asset-based approaches to study diverse ways of reasoning during debugging.

BRIAN R. BELLAND is associate professor of Educational Psychology at Pennsylvania State University, University Park. He uses a variety of quantitative, qualitative, and mixed methods approaches to study scaffolding in K–12 and higher education settings. He also conducts meta-analyses on STEM education.

AFAF BAABDULLAH is a doctoral candidate in the Learning, Design, and Technology program at Pennsylvania State University, University Park. Her research interests are in metacognitive collaborative learning and computing education. She is faculty with the Department of Curriculum and Instruction, King Saud University, Riyadh, Saudi Arabia.

EUNSEO LEE is a doctoral student in the Educational Psychology program at Pennsylvania State University, University Park. She is interested in measurement in problem-based learning contexts.

EMRE DINÇ is a doctoral student in the Learning, Design, and Technology program at Pennsylvania State University, University Park. His research interests are in teacher education for computing education.

ANNA Y. ZHANG is a doctoral student in the Educational Psychology program at Pennsylvania State University, University Park. Her research interests are in scaffolding design and machine learning.