Original Research Article

Asian Journal for Mathematics Education
2022, Vol. 1(4) 475-493

DOI: 10.1177/27527263221139869
journals.sagepub.com/home/mea

Differentiating between counterexamples for supporting students' algorithmic thinking

John G. Tupouniua

Abstract

This study examines the use of counterexamples for supporting the development of students' algorithmic thinking. Working from the premise that some counterexamples are more effective than others for the development of generalized algorithms, the study proposes distinctions between counterexamples in relation to the iterative refinement of student-invented algorithms. Furthermore, the study identifies some factors that may influence differences among counterexamples. Using task-based interviews, data were collected from 23 undergraduate students working in pairs ($n = 8$) and individually ($n = 7$) on three algorithmatizing tasks. From a thematic analysis of the data, two illustrative cases are presented to show how and why different counterexamples might bring about particular revisions in students' algorithms. The two illustrative cases highlight two types of counterexamples—*set-of-instructions-changing* (*SoI-changing*) and *domain-of-validity-narrowing* (*DoV-narrowing*)—and their influencing factors. Implications of the findings are discussed with respect to existing literature, further research, and teaching.

Keywords

algorithmic thinking, counterexamples, domain of validity, student-invented algorithms

Date received: 27 June 2022; accepted: 30 October 2022

1. Introduction

Recent studies have argued for the need to develop students' algorithmic thinking amid global changes in the mathematics education curricula and society more generally (Bocconi et al., 2016; Hart & Sandefur, 2018; Stephens & Kadijevich, 2019). Algorithmic thinking refers to the thinking involved in creating, testing, and revising an algorithm (e.g., Lehmann, 2021; Schute et al., 2017; Stephens & Kadijevich, 2019). One way to promote students' algorithmic thinking is by getting students to work on tasks that require students to engage in the iterative process of creating, testing, and

Institute of Education, Massey University, Auckland, New Zealand

Corresponding Author:
John G. Tupouniua, Institute of Education, Massey University, Auckland, New Zealand.
Email: j.g.tupouniua@massey.ac.nz

refining their own, student-invented, algorithms (Cai et al., 1998; Hart & Martin, 2018; Tupouniua, 2018). Furthermore, because the problems in these tasks are relatively novel, students are almost always not able to solve them by simply recalling an algorithm that they had previously learned.

The process of testing and refining an algorithm involves the use of counterexamples—problems for which an algorithm is expected to produce the correct solution(s), but does not (e.g., Clarke & Veith, 2003; Tupouniua, 2020a). The rationale behind the use of counterexamples is to help students realize that their current algorithm does not produce the correct answer for all the problems for which it is expected to produce the correct answer. In turn, this realization should motivate students to revise their current algorithm. Hence, a key measure of an algorithm's utility is the extent to which it is a *generalized algorithm*—one that produces correct solutions for a general class of problems. Yet, studies have found that when students are presented with counterexamples to their algorithm, or their ideas (more generally), they do not always conduct revisions that result in a generalized algorithm/idea (e.g., De Bock et al., 2002; Larsen & Zandieh, 2008; Moala et al., 2019; Tupouniua, 2020a). Research suggests that counterexamples which may be mathematically similar can be pedagogically different (Peled & Zaslavsky, 1997; Zazkis & Chernoff, 2008). That is, although an example can be determined a priori to be a counterexample—satisfying the conditions of a conjecture but violating the conclusion—there is no guarantee that the learner will recognize it as a counterexample.

The overarching goal of this study is to identify some differences between counterexamples in relation to the iterative refinement of student-invented algorithms, and some factors that may influence these differences. These distinctions could potentially help educators in choosing effective counterexamples to advance students' algorithms. Accordingly, the research question has two parts: What are some differences between counterexamples with respect to the revisions that they motivate in student-invented algorithms? And what are some factors that influence these differences? Toward providing some answers to the research question, two illustrative cases are presented in which students create and refine their own algorithms. The two cases highlight two different types of counterexamples that emerged in the students' activity plus some conditions that may have influenced their emergence. Then, the two types of counterexamples are discussed with respect to existing literature, further research, and teaching.

2. Background literature: Counterexamples in mathematics education

The role of counterexamples in learning mathematics has been well established as a means of occasioning cognitive conflict in students' mathematical thinking. A cognitive conflict is invoked when learners recognize an inconsistency in their ways of thinking. The resolution of this conflict is often equated to conceptual growth (e.g., Klymchuk 2010; Larsen & Zandieh, 2008; Peled & Zaslavsky, 1997; Zazkis & Chernoff, 2008). Instruction around counterexamples has also risen in prominence, mainly due to past and present reforms in mathematics education seeking to integrate the topic of mathematical proof and argumentation into all areas of the curricula (Nardi & Knuth, 2017; Stylianides et al., 2016). Consequently, a large portion of the existing mathematics education literature pertaining to the use of counterexamples relate to the topics of proof and argumentation. Some studies (e.g., Weber, 2009) have focused on examining how students can either generate counterexamples for the sake of disproving a conjecture or show that no counterexamples exist to prove the validity of a conjecture. Other studies have used existing, or created new, frameworks to analyze students' proofs—for instance, which kinds of counterexamples students discover, how students respond when confronted with counterexamples, and how students revise conjectures and proofs in response to counterexamples (Balacheff, 1991; Larsen & Zandieh, 2008; Tupouniua, 2020a; Zazkis & Chernoff, 2008). Moreover, others have focused on using counterexamples as

tools for teaching specific mathematics content, such as calculus and probability (e.g., Klymchuk, 2010; Klymchuk & Kachapova, 2012; Mason & Klymchuk, 2009).

The present study seeks to contribute to the existing literature by focusing on using counterexamples to support the development of students' algorithmic thinking—a line of research for which very little research currently exists. Specifically, the study builds on past studies that suggest that counterexamples which may be mathematically similar can be pedagogically different (e.g., Zazkis & Chernoff, 2006, 2008; Zazkis et al., 2008). Zazkis and Chernoff (2006, 2008) discussed the effect of different types of counterexamples on students' ways of thinking. Some counterexamples were dismissed and regarded by students as exceptions that did not affect the validity of their ways of thinking. But other counterexamples had more convincing power. Zazkis and Chernoff (2008) suggested that some counterexamples, *pivotal examples*, "[create] a turning point in the learner's cognitive perception or in his or her problem solving approaches" (p. 197). That is, pivotal examples help learners realize that a different way of thinking is needed because their current way of thinking is not universally valid. Other counterexamples, *bridging examples,* "[serve] as a bridge from the learner's initial (naïve, incorrect or incomplete conceptions) toward appropriate mathematical conceptions" (Zazkis & Chernoff, 2008, p. 197). In other words, a bridging example is a pivotal example, with the additional characteristic of helping learners obtain a universally valid way of thinking.

Interpreting Zazkis and Chernoff's constructs in the context of student-invented algorithms, the present study argues that these two counterexamples are relevant only with respect to a specific learning goal—supporting learners to create algorithms that solve a specific and fixed general class of problems (e.g., an algorithm that correctly determines which fraction in any pair of fractions is larger). So, when students for instance create a partially correct algorithm, Zazkis and Chernoff's goal, as inferred from their studies, would be to shift students away from the partially correct algorithm, and to attain an algorithm that correctly solves all the problems in the specified general class of problems. However, this is not the only desirable learning goal. When students create partially correct algorithms, a valuable goal is to help students identify the set of problems for which their current algorithm can correctly solve. Such a goal aligns with Sierpinska's (1994) argument that conceptual understanding should not be perceived as merely switching from a faulty way of thinking to a universally true way of thinking, but rather as "changing the status of these [initial ways of thinking] to 'one possible way of seeing things', or 'a locally valid way of thinking'" (p. 125). In other words, conceptual understanding involves becoming aware of the scope of applicability of different ways of thinking. Hence, the act of regarding counterexamples as exceptions can be viewed more positively as part of the process of identifying the scope of applicability of a way of thinking. Also, the counterexamples that were dismissed and regarded as exceptions from the perspective of past studies (Zazkis & Chernoff, 2006, 2008; Zazkis et al., 2008) are in fact significant. Moreover, previous research (e.g., Moala et al., 2019; Tupouniua, 2020a) found that students' activity on algorithmatizing tasks is non-linear in the sense that students may begin by limiting the scope of applicability of their algorithm, then proposing a different algorithm which they claim would be more suitable for their revised scope of applicability, then limiting (or expanding) the scope of applicability of their new algorithm, then changing the algorithm, or further limiting/expanding the scope of applicability, and so forth.

The limitations of Zazkis & Chernoff's *pivotal* and *bridging* examples, plus the non-linearity of students' algorithmatizing activity point to two things: (1) a need to revisit and differentiate between counterexamples in situations where students initially propose an algorithm that is partially correct, particularly in situations where the goal is not simply to attain a universally valid algorithm; and (2) a lens that affords tracking, and differentiating between, the non-linear changes that students implement on their algorithms. The first point is essentially the goal of this study. Toward addressing the second point, the next section introduces the notion of a *domain of validity* (DoV) and redefines the notions of student-invented algorithms and counterexamples (for student-invented algorithms).

3. DoV and student-invented algorithms

Duroux (1982, as cited in Brousseau, 1997) referred to the situations within which a way of thinking functions correctly as its *DoV*. Reiterating Sierpinska's (1994) argument, one could say that conceptual understanding entails becoming aware of the domains of validity of different ways of thinking. Using the notion of DoV, a student-invented algorithm can be viewed as comprising two parts: a *set of instructions* (SoI) that the student believes will enable one to solve specific problems; and *a* DoV —a set of problems, for which the set of instructions would, according to the student, yield correct solutions. Note, the DoV of a student-invented algorithm need not equate to the algorithm's normatively correct DoV (i.e., the entire set of problems for which the algorithm would yield correct solutions). Then, a *counterexample to a student-invented algorithm* can be redefined as a problem within the proposed DoV, for which the set of instructions does not produce the correct solution.

Given the foregoing conceptualization of a student-invented algorithm, one can say that an algorithm is *absolutely correct* if the set of instructions produces the correct solutions for all the problems within the corresponding DoV. An algorithm is *absolutely incorrect* if the set of instructions does not produce the correct solution for any of the problems within the DoV; and *partially (in)correct* if the set of instructions produces the correct answer for some, but not all, the problems within the DoV. To illustrate these concepts, suppose the following algorithm for subtraction is proposed—SoI: "subtract the smaller digit from the corresponding larger digit"; DoV: "all problems of form *a–b,* where *a* and *b* are any two positive integers." This algorithm is partially correct because it produces the correct answer for, say, 37–15, but not for, say, 37–18 (a counterexample). Changing the DoV of the algorithm to "all problems of the form *a–b*, where *a* and *b* are any two positive integers such that every digit in *a* is larger than its corresponding digit in *b*"—for example, $a = 3485$ and $b = 2361$—would make it absolutely correct. Similarly, changing the DoV to "all problems of form *a–b*, where *a* and *b* are any two positive integers such that every digit in *a* is less than its corresponding digit in *b*" (e.g., $a = 154$ and $b = 387$) would make the algorithm absolutely incorrect.

Overall, the re-conceptualizations provided in this section suggest that changes to student-invented algorithms could either be in terms of their SoI, or their DoV, or a combination of both. Additionally, progress in students' algorithms as per the different types of changes can be discussed with respect to the different types of (in)correctness rather than solely with respect to a fixed general set of problems.

4. Methods

This section begins with an outline of the larger study from which the two illustrative cases were taken. Then, specific information about the two illustrative cases, and how they were chosen from the larger dataset are explained, followed by a description of the data analysis methods.

4.1 The larger study

The data were collected as part of a self-designed research project that explores different aspects of students' algorithmic thinking. All participants of the project ($n = 23$) were recruited from different tertiary institutions in New Zealand, at different stages of their undergraduate degrees, and enrolled in at least one course in mathematics. Data were collected using task-based interviews (Maher & Sigley, 2020). The 23 participants were divided into eight pairs, and seven individuals. Students working in pairs had similar mathematical backgrounds. Each pair/individual worked on a set of three tasks, 45 min per task, with no more than one task per day. After every task, a 15-min follow-up interview was conducted, in which participants were asked to clarify parts of their work that were unclear to the researcher. A total of 45 task-based interview sessions took place, all of which were video recorded.

The three tasks were *algorithmatizing tasks* (Marrongelle, 2007; Tupouniua, 2018)—tasks in which the goal is to articulate an algorithm that would enable one to solve a given problem. Algorithmatizing tasks require students to engage in multiple iterations of creating, testing, and revising their algorithm with respect to counterexamples. Past studies have shown that algorithmatizing tasks are useful for eliciting students' algorithmic thinking processes and providing educators with ideas for enhancing students' algorithmic thinking (Cai et al., 1998; Marrongelle, 2007; Rasmussen et al., 2009; Moala et al., 2019; Tupouniua, 2020a). The three tasks are briefly summarized in Table 1 below.

The topics of all the tasks were chosen so that they were familiar and easy-to-access for participants. This decision meant that students spent more time creating and refining their algorithms rather than trying to understand the problem. Furthermore, the tasks were designed to increase the likelihood that creating an *absolutely correct* algorithm would not occur immediately, but rather to initially elicit partially (in)correct algorithms. Thus, for example, in *Comparing fractions* students needed to create a "relatively new" algorithm. The tasks were pilot tested on six students who had similar mathematical backgrounds to the research participants. Four of the pilot students worked in pairs, while two worked individually on all three tasks. The pilot tests led to minor revisions of the tasks, mainly in terms of clarity in the instructions and the problems that were offered as counterexamples to motivate revisions in the students' algorithms. From the pilot tests, it was also noted that for the purpose of ascertaining the DoVs of students' algorithms, students were to be constantly reminded to explicitly state what set of problems they thought their algorithm would correctly solve. Despite the pilot tests and preparation of counterexamples, it is important to reiterate that whether or not a problem is a counterexample to a student can often only be determined after the fact (Zazkis & Chernoff, 2008). Moreover, it was decided that during the sessions, the interviewers would not only offer problems that they knew were counterexamples to the students' algorithms but also offer problems that they knew were *not* counterexamples. This decision reflected the fact that what interviewers might consider to be counterexample might not be a counterexample to the students and vice-versa.

4.2 The two illustrative cases

4.2.1 How the two cases were chosen. Not all 45 task sessions were relevant with respect to the specific goal of this study—to revisit and differentiate between counterexamples in situations where

Table 1. Summary of Tasks.

Title	Description
Comparing fractions	Taken from Zazkis & Chernoff (2008), the task invites students to create a relatively new algorithm for determining which of two simple fractions (e.g., 1/2 vs 2/3) is larger. A "relatively new" algorithm is one which the students had not previously learned or could easily recall.
The shuttle replay problem	Taken from Tupouniua (2020b). Set in the context of a shuttle-relay run by four students. In each set of the relay, each student runs a distance equivalent to multiples of 25 meters. For example, in the first set the four students run 25, 50, 75, and 100 meters, respectively; the second set: 125, 150, 175, and 200 meters. The students are asked to create an algorithm for finding the total distance covered after any number of sets.
Expanding binomials	The task invites students to create an algorithm for expanding $(a+b)^n$ where n is any natural number. For each session in which this task was used, the value of n was varied depending on the mathematical background of the students. Some students worked on specific cases (e.g., $(a+b)^2$, $(a+b)^5$) while others worked on the general case $(a+b)^n$.

students initially propose an algorithm that is partially correct. Of the 45 cases, none of them were ones where the students began with an absolutely incorrect algorithm; eight were ones where the students found an absolutely correct algorithm on the first attempt (seven from *The shuttle relay* task, and one from *Expanding binomials*); and 37 were ones where the students began with a partially correct algorithm. Out of these 37 cases, the options for the two illustrative cases were narrowed to ones that involved multiple iterations of revisions of students' algorithms, and within which the two featured types of counterexamples that emerged in the analysis were evident. This decision resulted in 31 cases, a large majority of which were from *Comparing fractions* (15) and *Expanding binomials* (14). Accordingly, one case corresponding to each of these tasks was chosen—one from a pair of students, and one from an individual.

4.2.2 Case 1. At the time of data collection, the participants of the first case—Kelly and Beth (pseudonyms)—were enrolled in a pre-undergraduate bridging course, at a large university in New Zealand, covering selected topics in algebra. Kelly and Beth knew each other well and based on conversations we had about mathematics, both students had positive dispositions about mathematics. Case 1 is taken from Beth and Kelly's work on *Comparing fractions.* Prior to starting the task, I discussed algorithms for comparing two fractions with Kelly and Beth and found out that they knew the following strategies: (1) the common denominator strategy—converting fractions into fractions with a common denominator and then comparing; (2) subtracting one fraction from the other ($a-b$); if $a-b$ was positive then a was larger, and if negative, then b was larger; (3) converting fractions into decimals/percentages and comparing them. We discussed that while all these algorithms were valid, they needed to devise a different algorithm.

4.2.3 Case 2. The participant in the second case, Ned (pseudonym) was at the time of data collection enrolled in a predegree course in algebra at a New Zealand tertiary institution. Ned's aim was to become a secondary school mathematics teacher. Ned had taken 5 years away from studies after he completed secondary school and had returned to complete his undergraduate degree. Ned stated that mathematics was taught procedurally at his secondary school, and because of that he did not remember most of what he was taught. However, Ned acknowledged that since returning to study, he had developed a keen interest in mathematics, and was very enthusiastic about "upgrading his maths content knowledge." Case 2 is taken from Ned's work on *Expanding binomials*. After a brief conversation about what he had learned in his current algebra course, I found that Ned knew how to expand and simplify linear expressions easily, both single and multivariable, but did not know how to expand expressions involving exponents greater than one. In this session, Ned grappled with devising an algorithm for expanding $(a+b)^2$.

4.3 Data analysis

The results reported in this paper were obtained via thematic analysis (Braun & Clarke, 2012) of the data, taking place in three main stages. At the first stage, I read the transcripts several times and used the constructs described in the conceptual framework section to identify and code relevant transcript excerpts, corresponding to moments in which: (a) the students created an algorithm; (b) I offered a counterexample, or the participants generated a counterexample; and (c) the students responded to the counterexample and revised their algorithm in terms of the set of instructions and/or DoV. In the second stage of the analysis, I examined the excerpts identified in the first stage and classified the counterexamples according to similarities in the corresponding revisions that the students conducted. This classification led to two main types of counterexamples (*SoI-changing* and *DoV-narrowing*). A third type was also identified, but there was very

little evidence to justify its emergence. Then, in the third stage I examined all the scenarios corresponding to each of the two types of counterexamples, looking for factors that may have influenced why a counterexample motivated a particular type of revision. I identified these *factors* by examining the data excerpts pertaining to each of the two types of counterexamples and noticing similarities in the excerpts pertaining to each type (e.g., similarities in students' algorithms, similarities in the counterexamples that students were given or had generated, similarities in students' reasons for revising their algorithm). Then, the similarities I noticed for each of the two types of counterexamples were compared. For the sake of differentiating between the two types of counterexamples, the factors highlighted in this paper are those that were unique to each type.

The reliability of the analysis and results were checked in three ways. Firstly, interpretations of the counterexamples and the influencing factors underwent multiple cycles of revision to ensure that they were consistent across the relevant dataset. Secondly, interpretations of students' work, especially regarding why/how they revised their algorithm was shared with the participants for feedback. Students were informed to advise me if there was anything with which they disagreed. I did not receive any disagreements from students. Lastly, results were given to three colleagues for feedback. From colleagues' feedback, there were no inconsistencies in terms of the two main types of counterexamples, or the influencing factors. However, there were some differences in terms of how colleagues thought the two types of counterexamples were related to the counterexample types proposed by Zazkis and Chernoff (2008). These differences were considered, and interpretations were refined until all colleagues agreed.

5. Case 1

Two chronological episodes from Beth and Kelly's work on *Comparing fractions* are presented. Each episode begins with data excerpts followed by an analysis of the development of their algorithm (summarized in Tables 2 and 3, respectively). The two main types of counterexamples that emerged from the analysis—*SoI-changing counterexamples* and *DoV-narrowing counterexamples*—are introduced together with some explanations regarding the factors that influenced their emergence. As the names of these counterexamples suggest, the former is a counterexample that motivates a change in the set of instructions of the algorithm, while the latter results in a narrowing of the DoV of the algorithm.

Table 2. Summary of Algorithms from Episode 1 (Case 1).

Algorithm name	Set of instructions	Problem(s) on which the SoI was validated	Domain of validity	Counterexample and type
Algorithm 1	For each fraction, subtract the numerator from the denominator. The fraction corresponding to the smaller difference is the larger fraction.	2/3 vs 2/4	All pairs of simple fractions	2/6 vs 3/8 (set-of-instructions changing [SoI-changing])
Algorithm 2	For each fraction, multiply the numerator and the denominator. The fraction that has the bigger product is the larger fraction.	4/5 vs 3/5 2/6 vs 3/8	All pairs of simple fractions	N/A

Table 3. Summary of Algorithms from Episode 2 (Case 1).

Algorithm name	Set of instructions	Problem(s) on which the SoI was validated	Domain of validity	Counterexample (Type)
Algorithm 2	For each fraction, multiply the numerator and the denominator. The fraction that has the bigger product is the larger fraction.	2/6 vs 3/8	All pairs of simple fractions	1/3 vs 2/6 (DoV-narrowing)
Algorithm 3	For each fraction, multiply the numerator and the denominator. The fraction that has the bigger product is the larger fraction.	8/9 vs 9/10 21/49 vs 3/7	All pairs of simple fractions that are not equal	19/23 vs 23/29 (SoI-changing)

SoI-changing = set-of-instructions changing; DoV-narrowing = domain-of-validity-narrowing.

5.1 Episode 1 (Case 1)

The first pair of fractions given to Beth and Kelly is: 3/8 vs 2/6. After a moment of silence, Kelly says:

1. Kelly: I remember something from … was like you subtract the top from the bottom, and then the one that is closer to zero is the larger [fraction].
2. Beth: What? I don't get it!
3. Kelly: You take away the top number from the bottom number for both …
4. Beth: So, subtract the numerator from the denominator … oh I get it. And then whatever fraction has less … or closer together is the bigger fraction … like two over three [2/3] and two over four [2/4], 2/3 is bigger because it's [sic] closer to each other. Oh yeah, I remember that too.
5. Kelly: Yeah so if we use that for that [pointing at 3/8 vs 2/6] … it means … that [pointing at 3/8] is five and that one [pointing at 2/6] is 4. So, 2/6 is bigger.
6. Researcher: Can you use one of the other three strategies to check that? [brief period of silence ensues].
7. Beth: Ohhhh, it's wrong.
8. Kelly: Why?
9. Beth: Because 3/8 is 18/48 and 2/6 is 16/48. So, 3/8 is bigger than 2/6.
10. Kelly: And if you subtract 3/8 from 2/6 [puts numbers into calculator] you get a negative number, so 3/8 is obviously bigger! [period of silence ensues …].
11. Beth: How about … I think I got … we can times [multiply] the numerator and the denominator, then whatever number is bigger is the smaller fraction.
12. Kelly: Smaller? Or bigger?
13. Beth: What?
14. Kelly: The fraction with the bigger number is smaller, or bigger?
15. Beth: Ummm … [looks at her piece of paper] It's bigger …
16. Kelly: [Testing the new algorithm on 4/5 vs 3/5] Ok yeah … I think it's right.
17. Researcher: Can you test it on 3/8 vs 2/6 please?
18. Kelly: It works. And if we check it [uses the calculator, subtracting 2/6 from 3/8] it's right!

5.2 Analysis of Episode 1 (Case 1)

At the beginning of this episode Beth and Kelly created Algorithm 1, and interestingly, they first validated their algorithm on 2/3 vs 2/4, which they generated, rather than on the pair I had given them—2/6 vs 3/8 (see line 4). I then attempted to draw their attention (line 6) to how Algorithm 1 was partially (in) correct by asking them to use one of the other known algorithms to check which of 2/6 and 3/8 was larger. This led them to realize that 2/6 vs 3/8 was a counterexample to Algorithm 1 (lines 7–10).

In lines 11−15, one sees that immediately after recognizing that Algorithm 1 did not produce the correct result for 2/6 vs 3/8, Beth and Kelly modified their SoI. As such, in this situation 2/6 vs 3/8 can be thought of as an *SoI-changing counterexample*. This outcome differed from what I had expected (i.e., narrowing the DoV). Note, in Zazkis and Chernoff (2008), the student, Tanya, constantly narrowed the DoV of her initial algorithm, which was equivalent to Beth and Kelly's Algorithm 1, without modifying the set of instructions throughout the entirety of her work. Hence, one wonders about what might have led to a revision of the SoI as opposed to a narrowing of the DoV. I hypothesize that a change in the DoV occurred in Tanya's case (Zazkis & Chernoff, 2008) because the counterexamples she was given were ones that were relatively easy for her to categorize (e.g., 1/2 vs 2/4 was easily categorized as a representative of a pair of equivalent fractions). This in turn meant that it was easy for Tanya to narrow the DoV (e.g., from "all fractions" to "all non-equivalent pairs of fractions") by removing from the current DoV a class of fractions of which the counterexample was characteristic. In Beth and Kelly's case, however, 2/6 vs 3/8 was not as easy for them to categorize, and therefore not easy to narrow the DoV. This hypothesis is explored further in the next episode.

5.3 Episode 2 (Case 1)

After Kelly and Beth created and validated Algorithm 2, I say:

19. Researcher: Nice! What about this one [2/6 vs 1/3]? [They test Algorithm 2 on 2/6 vs 1/3, yielding 2/6 > 1/3. They then apply the common-denominator strategy and find that the two fractions are equal].
20. Researcher: What did you find?
21. Kelly: I should have known they were equal [both students laugh].
22. Researcher: Oh okay, how so?
23. Beth: It's wrong … because our strategy says that 2/6 is bigger [than 1/3], but they're actually equal … one-third is just two-sixths.
24. Kelly: Maybe it [Algorithm 2] works if the fractions are not the same … like not equal?
25. Beth: Um, yeah maybe … yeah … I think so too …cause that one was equal [1/3 vs 2/6].
26. Researcher: OK, so that algorithm [Algorithm 2] doesn't work on equal fractions … how about 9/10 vs 8/9?
27. Beth: So, 9/10 is 90, and 8/9 is … what's 8 times 9?
28. Kelly: What's eight times nine? Just use the calculator.
29. Beth: Seventy-two … okay so 9/10 is bigger. [Kelly validates the result using the common denominator strategy].
30. Beth: So, it works … our algorithm [Algorithm 2] works if the two fractions are not … the same … equal.
31. Researcher: Great. What about 3/7 vs 21/49. [The two students apply their algorithm on 3/7 vs 21/49].
32. Kelly: 21/49 is bigger … yeah, our strategy says this [21/49] is larger.
33. Beth: But … if we use … check it … they're exactly the same.
34. Kelly: What? [looks at Beth's piece of paper] Oh yeah [silence ensues]. Ohhh … yeah yeah that just means … It's OK because we said it works if the two fractions aren't equal.

35. Beth: Did we say that?
36. Kelly: Yeah, we did, it works if the fractions are not equal … like 1/3 and 2/6. And 3/7 and 21/49 are equal.
37. Beth: Ohhh, yeah.
38. Researcher: Okay, so it works if they aren't equal. Okay … how about 19/23 vs 23/29? [They apply Algorithm 2 on 23/29 vs 19/23].
39. Beth: Umm … 23/29 is bigger than 19/23.
40. Researcher: OK … can you check that?
41. Kelly: Yeah, I'm checking it now.
42. Kelly: So … I've got 19/23 is bigger than 23/29.
43. Beth: Me too.
44. Researcher: What does that mean in terms of your algorithm?
45. Beth: It's wrong. It works for some.
46. Kelly: It works if they're not equal.
47. Beth: Yeah, but it also doesn't work on some other ones, like this one [points at 19/23 vs 23/29].
48. Researcher: Some other fractions? Okay.
49. Kelly: Yeah, I don't know how to say these ones [pointing at 19/23 vs 23/29].
50. Beth: I think they have the same thing … difference … right? Like if you subtract top from bottom you get four and four. Wait, no that's not right … that's six [pointing at 23/29]. I think we need a new strategy.
51. Kelly: This strategy was good, but then we got to these ones [pints at 19/23 and 23/29] and it didn't work … I think you're right; we need another one. Maybe, like subtract top from bottom, or bottom from top and then the one that is closer to zero is smaller.
52. Beth: No, we already did that … Wasn't that the one we first tried? And that's wrong. Before, we already tried that, and it was wrong.

5.4 Analysis of Episode 2 (Case 1)

At the beginning of this episode Kelly and Beth recognized 1/3 vs 2/6 as a counterexample to Algorithm 2. Consequently, they responded by narrowing the DoV of Algorithm 2 (see lines 24–25). As such, in this situation, 1/3 vs 2/6 is a DoV-narrowing counterexample. More specifically, Kelly and Beth removed from Algorithm 2's DoV a class of fractions (i.e., equal fractions) of which they deemed 1/3 vs 2/6 an example. Here one sees that Kelly and Beth did not change the SoI of their algorithm as they did in Episode 1 for their first algorithm. This seems to support the hypothesis from Episode 1 regarding a potential difference between DoV-narrowing and SoI-changing counterexamples: 1/3 vs 2/6 was relatively easy for the students to categorize as belonging to a larger group of fractions (i.e., equal fractions), whereas 2/6 vs 3/8 from Episode 1 was not.

After acknowledging Algorithm 3, I gave Beth and Kelly two problems 8/9 vs 9/10 and 21/49 vs 3/7 (see lines 26–31), which were not counterexamples to Algorithm 3. As previously explained in Section 4, I intentionally offered the students problems that were not, from my perspective, counterexamples to their algorithm, to elicit students' responses to what I deemed as non-counterexamples. Beth and Kelly responded (lines 26–37) by noting that Algorithm 3 worked correctly on 8/9 vs 9/10, and 3/7 vs 21/49 did not exist within the scope of Algorithms 3's DoV, so neither of them were counterexamples. This further validated Beth and Kelly's claim that Algorithm 3 worked correctly on fractions which were not equal.

Finally, I presented the students with 19/23 vs 23/29 (line 38). I purposely chose this example because I conjectured that it would not be easy for Beth and Kelly to categorize (as opposed to the previous DoV-narrowing counterexample of 1/3 vs 2/6). Upon recognizing that it was a

counterexample, Beth and Kelly initially tried to categorize it, presumably in the hope that they could then narrow their DoV (see lines 45–50). However, Beth and Kelly were not able to categorize it, and thus considered changing their SoI (see lines 50–51).

6. Case 2

Two chronological episodes are presented here on Ned's work on *Expanding binomials*, specifically $(a+b)^2$. Each episode begins with data excerpts, followed by interpretations focusing on the development of Ned's algorithm in relation to the two types of counterexamples introduced in Case 1. Table 4 and Table 5 contain summaries of Ned's algorithms evident in Episode 1 and Episode 2, respectively.

6.1 Episode 1 (Case 2)

Ned reads the task instructions a couple of times, and then says:

1. Ned: This is one of those tricky ones aye? Like it makes you overcomplicate it. Problem solving 101, always try the simplest solution. If the computer is not working, turn it off and turn it back on [laughs].
2. Researcher: True! [A short period of silence ensues as Ned stares at the problem].
3. Ned: OK, for this I think you would just raise each of them [sic] power of two…. like square both of them and then add them.
4. Researcher: Sorry can you repeat that, please?
5. Ned: You square a, then square b … and then add them [writes down $(a+b)^2=a^2+b^2$].
6. Researcher: Okay. So that's your algorithm?
7. Ned: Yes.
8. Researcher: Can you test your algorithm … that algorithm on a equals two and b equals negative two?
9. Ned: [writes out $(2+-2)^2$] So that's just zero, because two plus negative two is zero, and zero raised to power two is zero.
10. Researcher: Okay that's right … now what value does your algorithm give you?
11. Ned: My algorithm? Oh [laughs] sorry I … [writes $2^2+-2^2=4+4=8$]. Hmmm … is that right? Why is that 8? Nah, that's correct … so I think … [silence ensues].
12. Ned: OK, so my algorithm is wrong, because [points at $(2+-2)^2$] should be zero, but I got eight. But … [extended period of silence].
13. Researcher: Can you tell me what you are thinking?

Table 4. Summary of Algorithms from Episode 1 (Case 2).

Algorithm name	Set of instructions	Problem(s) on which the SoI was validated	Domain of validity	Counterexample
Algorithm A	a^2+b^2	None	All pairs of a and b	$a=2$, $b=-2$ (DoV-narrowing)
Algorithm B	a^2+b^2	None	All pairs of a and b, such that $\|a\| \neq \|b\|$	$a=1$, $b=2$ (SoI-changing)
Algorithm C	a^2+b^2+4	$a=1$, $b=2$	All pairs of a and b, such that $\|a\| \neq \|b\|$	N/A

SoI-changing = set-of-instructions changing; DoV-narrowing = domain-of-validity-narrowing.

Table 5. Summary of Algorithms from Episode 2 (Case 2).

Algorithm name	Set of instructions	Problem(s) on which the SoI was validated	Domain of validity	Counterexample
Algorithm C	a^2+b^2+4	$a=1$, $b=2$	All pairs of a and b, such that $\|a\| \neq \|b\|$	$a=2$, $b=3$ (SoI-changing)
Algorithm D	a^2+b^2+ positive number	None	All pairs of a and b, such that $\|a\| \neq \|b\|$	$a=2$, $b=0$ (SoI-changing)
Algorithm E	a^2+b^2	$a=2$, $b=0$	All pairs of a and b	$a=1$, $b=2$ (DoV-narrowing)
Algorithm F	a^2+b^2	$a=2$, $b=0$	All pairs of a and b, such that one is positive and the other is zero	$a=0$, $b=0$ (neither DoV-narrowing, nor SoI-changing)
Algorithm G	a^2+b^2	N/A	All pairs of a and b, such that at least one of a and b is zero	N/A

SoI-changing = set-of-instructions changing; DoV-narrowing = domain-of-validity-narrowing.

14. Ned: I'm not thinking, I'm overthinking [laughs]. I'm thinking if they're not the same number then it should work. Like you see how you have two and negative two … or two and two … or negative one and one … then I think it'll work.
15. Researcher: Okay. Let's test that … What about if a equals one and b equals two … so they're not the same.
16. Ned: Yeah, not the same. So should work [writes out $(1+2)^2=9$ … then writes $1^2+2^2=5$]. Still don't [sic] work! What's wrong? What am I doing wrong? I think it's not just the two numbers each raised to two and then add `em. I think you need to add another number to them aye? Like for this one [points to $1^2+2^2=5$] you have to add four … and that works … maybe that's it, you square them and add them and then add four. I'll go with that.
17. Researcher: Okay, good. So, you've adjusted your strategy there. And you're saying that might work for all values of a and b?
18. Ned: Yes. Wait! Nah nah … what did I say before? I think it works when they're like not the same, like negative one and one, or two and two.
19. Researcher: Ah! OK, got you!

6.2 Analysis of Episode 1 (Case 2)

At the beginning of this episode, upon noticing that Algorithm A was partially (in)correct my goal was to try and direct Ned toward the normatively correct DoV for this SoI (i.e., all pairs of a and b, such that $a=0$ or $b=0$). Toward this goal, I presented Ned (line 8) with a counterexample, $a=2$ and $b=-2$. This ultimately, led to Ned recognizing $a=2$ and $b=-2$ as a counterexample to Algorithm A. As anticipated, Ned responded (line 14), by narrowing the DoV of Algorithm A. More specifically, Ned categorized the counterexample, $a=2$ and $b=-2$ as a pair of numbers with the same absolute value, and removed this category from his current DoV.

Subsequently, I presented Ned with a counterexample to Algorithm B: $a=1$ and $b=2$ (in line 15). Upon testing Algorithm B on this problem, and realizing that it did not produce the correct result (line 16), Ned created Algorithm C by changing the SoI, but maintaining the DoV, of Algorithm B. Thus, $a=1$ and $b=2$ was an SoI-changing counterexample in this situation. In summary, this episode seems to support the hypotheses from Beth and Kelly's case, about how counterexamples that are

"relatively easy to categorize" (e.g., $a = 2$ and $b = -2$) result in a narrowing of the DoV, while counterexamples (e.g., $a = 1$ and $b = 2$) that are not relatively easy to categorize lead to a changing of the SoI.

6.3 Episode 2 (Case 2)

After Ned creates Algorithm C (see lines 16–18 above), I say:

20. Researcher: Can you test your algorithm on a equals two and b equals three?
21. Ned: So, a equals two and b equals three. Okay my … that will be [writes $2^2 + 3^2 + 4 = 17$] … now, two plus three is five and five square is … oh no no … Five to the power of two is twenty … it's not seventeen! Five square is … sorry, twenty-five. This is fun but tricky. Okay so I think my thinking is correct but it's not add four … you need to add like … a positive number … so a square plus b square plus a positive number … but I don't know what that positive number is.
22. Researcher: OK [extended period of silence ensues]. What if I gave you a equals two and b equals zero?
23. Ned: So, the correct answer is four, because two plus zero is two, then to the power of two is four [he writes $(2 + 0)^2 = 2^2 + 0 +$ a positive number]. So, four plus a positive number if I use my algorithm.
24. Researcher: What does that give you …?
25. Ned: Wait a sec … so … it's just four … it should be four plus zeros [sic] … so maybe my first algorithm was correct … I just need to square both of them [a and b] and then plus [sic] them.
26. Researcher: So, you're saying that your initial algorithm was right?
27. Ned: Yeah … that first one works here, so maybe I tested it wrong the first time.
28. Researcher: Can you test your first algorithm on a equals one and b equals two?
29. Ned: [Writes $(1 + 2)^2 = 9$ and $1^2 + 2^2 = 5$] OK it doesn't work … Phew, I thought I was losing my mind [laughs]. But maybe that first one … it works for only some numbers aye … So what was that one it worked for? [Pointing at $a = 2$ and $b = 0$] I think one has to be greater than zero and the other one is zero. So, it doesn't work for a equals one and b equal [sic] two. But it works for like three and zero or zero and fifteen or billions [sic] and zero [laughs].
30. Researcher: How about if they're both zero?
31. Ned: If they're both zero? Oh!
32. Researcher: If like a is zero and b is zero, so neither of them is positive.
33. Ned: Ummm…. oh, yeah, that'll work too … cause both will be equal to zero … And I think it works even if one of them is negative and the other is zero, right? Like negative two and other one is zero, because that'll just be like if one was positive and the other was zero … because you're taking the negative and raising it to power two [sic], so it becomes positive.

6.4 Analysis of Episode 2 (Case 2)

At the beginning of this episode, I gave Ned $a = 2$ and $b = 3$, which he recognized as a counterexample to Algorithm C (line 21). Then, Ned modified the SoI and maintained the DoV of Algorithm C. As such, in this situation, $a = 2$ and $b = 3$ can be viewed as an SoI-changing counterexample, because it was not as easy for Ned to categorize unlike, say, $a = 2$ and $b = -2$ from Episode 1.

After Ned created Algorithm D, I noted that Ned had gone through two consecutive iterations of modifying his SoI. At this point, I had to make a decision about whether I wanted Ned to continue toward a correct SoI for his current DoV or revisit his original SoI (for Algorithm A) and work toward its normatively correct DoV. I chose the latter, because Ned's initial algorithm was partially correct

so I wanted him to try and identify the correct DoV of his initial SoI, rather than abandoning it. Thus, in line 22 I purposefully chose a counterexample, $a = 2$ and $b = 0$, that I anticipated would motivate Ned to reconsider the validity of his original SoI. Note, this problem is not only a counterexample to Algorithm D, but also an element of the correct DoV for Ned's original SoI (i.e., $a^2 + b^2$). Upon being given $a = 2$ and $b = 0$, Ned abandoned his current SoI and created Algorithm E, which was equivalent his initial algorithm (see lines 23–27).

Then, I immediately asked Ned (line 28) to consider $a = 1$ and $b = 2$, with the hope that it would motivate a narrowing of the DoV. However, in hindsight, I had overlooked the fact that in Episode 1 this exact counterexample had led to a change in Ned's SoI. Surprisingly, after recognizing that $a = 1$ and $b = 2$ was a counterexample to Algorithm E (line 29), Ned seemed to differentiate between the two most recent counterexamples he was given: $a = 2$ and $b = 0$; $a = 1$ and $b = 2$. Subsequently, Ned narrowed the DoV of Algorithm E and declared that his current SoI would work if one of a or b was positive and the other was zero. Ned's revision of the DoV in this situation surprised me because it contrasted what he had previously done in Episode 1. When Ned encountered $a = 1$ and $b = 2$ in Episode 1, he changed the SoI of Algorithm A, because as I had hypothesized, he did not find it easy to categorize this problem. However, this time around it was perhaps easier for Ned to categorize $a = 1$ and $b = 2$, because Ned had seen another concrete problem/example, $a = 2$ and $b = 0$, on which he knew his SoI functioned correctly, and with which he could compare $a = 1$ and $b = 2$. In Episode 1, such a concrete problem/example might not have been readily available to Ned. Therefore, the status of $a = 1$ and $b = 2$ had changed from an SoI-changing example (in Episode 1) to a DoV-changing example (in Episode 2).

Although Algorithm F (in line 29) was *absolutely correct*, in the hope that Ned would reach the complete normatively correct DoV for his SoI, I again offered another problem to him: $a = b = 0$. Ned responded by expanding the DoV of Algorithm F, thus producing Algorithm G. This problem does not belong to the two main types of counterexamples that emerged in the analysis, and thus potentially points to another type of counterexamples. However, very little data beyond this situation exist to justify its emergence in this study.

7. Summary and discussion

This section situates SoI-changing and DoV-narrowing counterexamples plus their influencing factors within relevant literature and highlights some implications for teaching and future research.

7.1 Two different types of counterexamples for supporting student-invented algorithms

For either of the two types of counterexamples that emerged in this study, the learner must first recognize it as a counterexample from his/her perspective — that is, the problem exists in his/her current DoV. Then, some counterexamples—*DoV-narrowing counterexamples*—motivate students to reduce the size of the DoV while maintaining the SoI of their current algorithm. Note, these counterexamples are ones that were regarded in past studies as ineffective because they were dismissed and viewed as exceptions (e.g., Zazkis & Chernoff, 2008). The present study found that DoV-narrowing counterexamples are ones that were relatively easy for students to categorize (e.g., equal fractions, numbers with the same absolute value). The question, then is: what makes a problem (example) "relatively easy for the student to categorize"? The concept of *personal (local and situated) example spaces* (e.g., Goldenberg & Mason, 2008; Sinclair et al., 2011; Watson & Mason, 2005) which Zazkis and Chernoff (2008) also drew on in their conceptualizations of pivotal and bridging examples, is useful for answering this question. According to Sinclair et al. (2011) any person, in any given situation, has a personal example space that "may contain central, obvious, examples that come to mind, triggered by a word or familiar context. To a learner these

may be isolated cases, while to a more experienced mathematician they may represent a class of related and reconstructible paradigmatic objects" (p. 292). Thus, in the context of students creating, testing, and revising algorithms, the students' personal example spaces include, but are not limited to, problems on which they think the algorithm would work, and problems on which they think the algorithm would not work. Along this line of thinking, there are two ways to describe the conditions that make a problem DoV-narrowing:

1. The problem is readily recognized by the learner as being of the same class as other examples that exist in the learner's personal example space (e.g., 1/3 and 2/6 is readily classified as *equivalent fractions*, because the learner has other examples of equivalent fractions in mind).
2. The problem can be readily distinguished by the learner from examples that exist in the learner's personal example space. For instance, in Case 2 of the present study, Ned had identified $a = 2$ and $b = 0$ as an example of a problem which he knew his algorithm could solve correctly. Subsequently, upon being given $a = 1$ and $b = 2$, Ned acknowledged it as a counterexample, and then distinguished $a = 1$ and $b = 2$ from $a = 2$ and $b = 0$, by categorizing $a = 1$, $b = 2$ as an example of problems in which neither a nor b is zero).

In addition to DoV-narrowing counterexamples, the study found that some other counterexamples—*SoI-changing counterexamples*—motivate students to revise the SoI of their current algorithm. These counterexamples, unlike DoV-narrowing counterexamples, are ones that were not relatively easy for students to categorize. Again, using the language of *example spaces*, a problem is an SoI-changing counterexample if there does not exist an example (or a group of examples) in the learner's personal example space to which the problem can be readily likened or distinguished. Note, Zazkis and Chernoff's (2008) *pivotal* and *bridging counterexamples* can be subsumed under the category of SoI-changing counterexamples. Specifically, a counterexample is *pivotal* if it motivates the student to change his/her SoI with respect to a fixed general class of problems (e.g., all pairs of fractions), and *bridging* if the change to the SoI results in an algorithm that can correctly solve all the problems in the general class. Here, one sees that pivotal and bridging examples pertain specifically to cases where students change their SoI while maintaining a DoV that is equivalent to the general class of problems.

For both SoI-changing and DoV-narrowing counterexamples, it is evident that the structure of any learner's personal example space (which includes the current DoV in conjunction with prior knowledge) is influential in terms of how they respond to the offered problem. Thus, despite this study's attempt to differentiate between counterexamples, it is important to note that the status of these examples may be different for different people. That is, what may be an SoI-changing example to one person may be a DoV-changing example to another. For instance, 2/6 vs 3/8 was SoI-changing for Kelly and Beth, but another student might have categorized it as a pair of fractions with numerators equal to 2 or 3, hence potentially making it a DoV-changing example. Moreover, as evident in Case 2, even within a single task, it is possible that a counterexample can change from SoI-changing to DoV-narrowing with respect to the same person. This can perhaps be attributed to the dynamic and constantly evolving structure of the learner's example space (Watson & Mason, 2005; Goldenberg & Mason, 2008; Sinclair et al., 2011).

The findings of this study contribute to a very small body of literature on the use of counterexamples specifically in relation to student-invented algorithms and algorithmic thinking. A phenomenon examined in past research is that when students recognize counterexamples to their algorithms, they do not always conduct revisions that result in generalized algorithms (De Bock et al., 2002; Larsen & Zandieh, 2008; Moala et al., 2019; Tupouniua, 2020a). A plausible explanation for this phenomenon is that some counterexamples are better than others with respect to supporting the development of generalized algorithms (Zazkis & Chernoff, 2008). Thus, the present study contributes to our

understanding of this phenomenon, by offering some differences between counterexamples in terms of the changes they may elicit in students' algorithms and the conditions that influence these differences. Furthermore, the study sheds some light on productive ways for educators to respond in the moment to the small revisions that students implement to their algorithms, and how they might offer appropriate counterexamples that help students build on these small revisions, rather than simply expecting students to abandon their partially (in)correct algorithms in favor of generalized algorithms. The following section briefly highlights some ways educators might use counterexamples to support the development of student-invented algorithms in the moment.

7.2 Suggestions for supporting the development of student-invented algorithms

Ideally, in any case, the teacher's goal is to support students work toward an absolutely correct algorithm (i.e., one for which the SoI would correctly solve all the problems within the corresponding DoV). This would involve students narrowing/expanding the DoV, altering the SoI, or a combination of the two. The suggestions below focus on cases in which the initial algorithm is partially (in)correct, since this was the scenario around which the present study revolved.

One way to support students with refining their algorithm is to offer them a DoV-narrowing problem. A DoV-narrowing problem may be needed when the DoV of the student's current algorithm is larger than the normatively correct DoV (e.g., $(a+b)^2 = a^2 + b^2$ for all real numbers a and b). Motivating students to narrow their algorithm's DoV is reminiscent of what Lakatos (1976) refers to as *exception barring*, a process in which one treats counterexamples as exceptions to a theorem, while trying to establish a safe DoV (Larsen & Zandieh, 2008). In other scenarios, it may be more useful to motivate a change in the SoI rather than refining the DoV because reaching an absolutely correct algorithm by way of narrowing the DoV, may be too difficult for the learner relative to their current mathematical knowledge. This scenario was evident in Beth and Kelly's case. However, if it is relatively easy for the student to reach the correct DoV, such as in Ned's case, then it may be worthwhile to support the student in attaining the correct DoV.

Finally, it is imperative for educators to remember that the status of any counterexample can change while students engage with a task. This change of status is influenced by the dynamic nature of the student's personal example space, which in turn can be influenced to some extent by the problems/examples the teacher offers throughout the session. Choosing appropriate counterexamples requires paying close attention to what students are doing in the moment.

7.3 Limitations and future directions

The limitations resulting from the conceptualization, design and context of this study serve as potential avenues for future research. Firstly, the definition of a *student-invented algorithm* in terms of SoI and DoV influenced the types of counterexamples found. Alternative definitions may result in the emergence of different types of counterexamples. Also, the study's specific goal of differentiating between counterexamples in situations where students initially propose partially correct algorithms, and educators are not merely interested in students developing universally valid algorithms, may have influenced the types of counterexamples found. Secondly, the participants of the study were from a single educational sector. Future research might explore the use of counterexamples for supporting student-invented algorithms with participants from educational sectors other than tertiary (i.e., primary, secondary). It would also be useful for future research to use algorithmatizing tasks in which the problem is less straightforward, both in terms of the mathematical topics and the layers of complexity within the instructions, than those of the tasks in this study. Thirdly, a limitation of the study is that the three tasks were ones that were more likely to elicit partially (in)correct algorithms rather than absolutely incorrect algorithms. Thus, future research could explore cases where

the student's initial algorithm is absolutely incorrect. Presumably, for cases in which students create absolutely incorrect algorithms, the goal would be to support students ultimately realize that no DoV exists for their SoI. Hence, DoV-narrowing examples may be needed. But will the influencing conditions be the same as the ones found in the present study? Lastly, this study rests on the premise that the DoV of students' algorithms are relatively apparent to the instructor and the students. The task-based interviews were designed in such a way that students would make their DoVs explicit. However, it is not too far-fetched to imagine situations in which students create an SoI but cannot ascertain a set of problems for which they think their SoI would work. For instance, in Ned's case another student may have proposed an initial algorithm of $a^2 + b^2$, but instead of claiming that it works for all pairs of a and b, they might claim that it works "for some [unspecified] values of a and b". Such a scenario is one that will resonate with many mathematics instructors. Exploring the role of counterexamples when the student's DoV is unspecified may provide further insights into supporting the development of students' algorithmic thinking.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

ORCID iD

John G. Tupouniua https://orcid.org/0000-0001-6541-4229

References

Balacheff, N. (1991). Treatment of refutations: Aspects of the complexity of a constructivist approach to mathematics learning. In E. von Glasersfeld (Ed.), *Radical constructivism in mathematics education* (pp. 89–110). Dordrecht, The Netherlands: Kluwer Academic Publishers. https://doi.org/10.1007/0-306-47201-5_5

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). *Developing computational thinking in compulsory education: Implications for policy and practice* (Report No. 68). European Union Commission, Joint Research Centre. https://doi.org/10.2791/792158

Braun, V., & Clarke, V. (2012). Thematic analysis. In H. Cooper, P. M. Camic, D. L. Long, A. T. Panter, D. Rindskopf, & K. J. Sher (Eds.), *APA handbook of research methods in psychology, Vol. 2: Research designs: Quantitative, qualitative, neuropsychological, and biological* (pp. 57–71). Washington, DC: American Psychological Association. https://doi.org/10.1037/13620-004

Brousseau, G. (1997). *Theory of didactical situations in mathematics*. Dordrecht, The Netherlands: Kluwer. https://doi.org/10.1007/0-306-47211-2

Cai, J., Moyer, J. C., & Laughlin, C. (1998). Algorithms for solving non-routine mathematical problems. In L. J. Morrow, & M. J. Kenney (Eds.), *The teaching and learning of algorithms in school mathematics, 1998 yearbook* (pp. 218–229). Reston, VA: National Council of Teachers of Mathematics.

Clarke, E., & Veith, H. (2003). Counterexamples revisited: Principles, algorithms, applications. In N. Dershowitz (Ed.), *Verification: Theory and practice* (pp. 208–224). Berlin, Germany: Springer. https://doi.org/10.1007/978-3-540-39910-0_9

De Bock, D., & Van Dooren, W., D. Janssens, & L Verschaffel. (2002). Improper use of linear reasoning: An in-depth study of the nature and the irresistibility of secondary school students' errors. *Educational Studies in Mathematics*, *50*(3), 311–334. https://doi.org/10.1023/A:1021205413749

Goldenberg, P., & Mason, J. (2008). Shedding light on and with example spaces. *Educational Studies in Mathematics*, *69*(2), 183–194. https://doi.org/10.1007/s10649-008-9143-3

Hart, E. W, & Martin, W. G. (2018). Discrete mathematics is essential mathematics in a 21st century school curriculum. In E. W. Hart, & J. Sandefur (Eds.), *Teaching and learning discrete mathematics worldwide: Curriculum and research* (pp. 3–19). New York, NY: Springer Cham. https://doi.org/10.1007/978-3-319-70308-4_1

Hart, E. W, & Sandefur, J. (Eds.). (2018). *Teaching and learning discrete mathematics worldwide: Curriculum and research*. New York, NY: Springer Cham. https://doi.org/10.1007/978-3-319-70308-4

Klymchuk, S. (2010). *Counterexamples in calculus*. Washington, DC: MAA. https://doi.org/10.1090/clrm/034

Klymchuk, S., & Kachapova, F. (2012). Paradoxes and counterexamples in teaching and learning of probability at university. *International Journal of Mathematical Education in Science and Technology*, *43*(6), 803–811. https://doi.org/10.1080/0020739X.2011.633631

Lakatos, I. (1976). *Proofs and refutations*. Cambridge, UK: Cambridge University Press.

Larsen, S., & Zandieh, M. (2008). Proofs and refutations in the undergraduate mathematics classroom. *Educational Studies in Mathematics*, *67*(3), 205–216. https://doi.org/10.1007/s10649-007-9106-0

Lehmann, T. H. (2021). Making sense of algorithms in discrete mathematics. *International Journal of Science and Mathematics Education*, *20*(5), 1–21. https://doi.org/10.1007/s10763-021-10180-3

Maher, C. A., & Sigley, R. (2014). Task-based interviews in mathematics education. In S. Lerman (Ed.), *Encyclopedia of mathematics education* (pp. 579–582). Dordrecht, The Netherlands: Springer. https://doi.org/10.1007/978-94-007-4978-8_147

Marrongelle, K. (2007). The function of graphs and gestures in algorithmatization. *The Journal of Mathematical Behavior*, *26*(3), 211–229. https://doi.org/10.1016/j.jmathb.2007.09.005

Mason, J. H., & S Klymchuk. (2009). *Using counter-examples in calculus*. London, UK: Imperial College Press. https://doi.org/10.1142/p627

Moala, J. G., Yoon, C., & Kontorovich, I. (2019). Localized considerations and patching: Accounting for persistent attributes of an algorithm on a contextualized graph theory task. *The Journal of Mathematical Behavior*, *55*(1), 100704. https://doi.org/10.1016/j.jmathb.2019.04.003

Nardi, E., & Knuth, E. (2017). Changing classroom culture, curricula, and instruction for proof and proving: How amenable to scaling up, practicable for curricular integration, and capable of producing long-lasting effects are current interventions? *Educational Studies in Mathematics*, *96*(2), 267–274. https://doi.org/10.1007/s10649-017-9785-0

Peled, I., & Zaslavsky, O. (1997). Counter-examples that (only) prove and counter-examples that (also) explain. *Focus on Learning Problems in Mathematics*, *19*(3), 49–61. https://eric.ed.gov/?id=EJ558849

Rasmussen, C., Zandieh, M., King, K., & Teppo, A. (2009). Advancing mathematical activity: A practice-oriented view of advanced mathematical thinking. *Mathematical Thinking and Learning*, *7*(1), 51–73. https://doi.org/10.1207/s15327833mtl0701_4

Schute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*(1), 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Sierpinska, A. (1994). *Understanding in mathematics*. London, UK: Routledge. https://doi.org/10.4324/9780203454183

Sinclair, N., Watson, A., Zazkis, R., & Mason, J. (2011). The structuring of personal example spaces. *The Journal of Mathematical Behavior*, *30*(4), 291–303. https://doi.org/10.1016/j.jmathb.2011.04.001

Stephens, M., & D. M Kadijevich. (2019). Computational/algorithmic thinking. In S. Lerman (Ed.), *Encyclopedia of mathematics education* (pp. 483–451). Dordrecht, The Netherlands: Springer. https://doi.org/10.1007/978-3-319-77487-9_100044-1

Stylianides, A. J., K. N Bieda., & F Morselli. (2016). Proof and argumentation in mathematics education research. In A. Gutiérrez, G. C. Leder, & P. Boero (Eds.), *The second handbook of research on the Psychology of mathematics education* (pp. 315–351). Rotterdam, The Netherlands: Brill Sense. https://doi.org/10.1007/978-94-6300-561-6_9

Tupouniua, J. G. (2018). Exploring mechanisms by which student invented algorithms in mathematics emerge [Doctoral dissertation, The University of Auckland]. UoA Research Repository. http://hdl.handle.net/2292/47350

Tupouniua, J. G. (2020a). Explicating how students revise their algorithms in response to counterexamples: Building on small nuanced gains. *International Journal of Mathematical Education in Science and Technology*, *53*(7), 1711–1732. https://doi.org/10.1080/0020739X.2020.1837402

Tupouniua, J. G. (2020b). Finding correct solution(s) ⇏ creating correct algorithm(s): Shedding more light on how and why. *The Mathematician Educator*, *1*(2), 102–121. https://ame.org.sg/2021/01/06/tme2020-vol-1-no-2-pp-102-121/

Watson, A, & Mason, J. (2005). *Mathematics as a constructive activity: Learners generating examples*. Mahwah, NJ: Erlbaum. https://doi.org/10.4324/9781410613714

Weber, K. (2009). How syntactic reasoners can develop understanding, evaluate conjectures, and generate counterexamples in advanced mathematics. *The Journal of Mathematical Behavior*, *28*(2–3), 200–208. https://doi.org/10.1016/j.jmathb.2009.08.001

Zazkis, R., & Chernoff, E. (2006). Cognitive conflict and its resolution via pivotal/bridging example. In J. Novotná, H. Moraová, M. Krátká, & N. Stehlíková (Eds.), *Proceedings of the 30th Conference of the International Group for the Psychology of Mathematics Education* (Vol. 5, pp. 465– 472). Prague, Czech Republic: PME. https://www.researchgate.net/publication/238734117

Zazkis, R., & Chernoff, E. (2008). What makes a counterexample exemplary? *Educational Studies in Mathematics*, *68*(3), 195–208. https://doi.org/10.1007/s10649-007-9110-4

Zazkis, R., Liljedahl, P., & Chernoff, E. J. (2008). The role of examples in forming and refuting generalizations. *ZDM—The International Journal on Mathematics Education*, *40*(1), 131–141. https://doi.org/10.1007/s11858-007-0065-9

Author biography

John G. Tupouniua is currently a Lecturer in Mathematics Education at the Massey University, New Zealand. His current primary area of research is algorithmic thinking. His other research interests include making mathematics accessible to all learners and debunking myths about mathematics (and mathematics education). Also published as Moala, John Griffith; Moala, J., Moala, J. G.