Check for updates

Journal of Computer Assisted Learning

WILEY

Journal of Computer Assisted Learning

ORIGINAL ARTICLE OPEN ACCESS

A Learner-Centred Exploration of Teachers' Solution Pathways in K-12 Programming-Based Mathematical Problem-Solving

Huiyan Ye[1] | Biyao Liang[2] | Oi-Lam Ng[1]

[1]Department of Curriculum and Instruction, Faculty of Education, The Chinese University of Hong Kong, Hong Kong, China | [2]Faculty of Education, The University of Hong Kong, Hong Kong, China

Correspondence: Oi-Lam Ng (oilamn@cuhk.edu.hk)

Received: 3 September 2024 | **Revised:** 14 June 2025 | **Accepted:** 12 July 2025

Funding: This study was fully supported by the Hong Kong Research Grants Council, General Research Fund (Ref No. 14603720).

Keywords: computational thinking | mathematical thinking | mathematics education | problem solving | programming | solution pathways

ABSTRACT

Background: Empirical studies have revealed students' development of computational thinking (CT) and mathematical thinking (MT) during programming-based mathematical problem-solving, highlighting specific CT concepts or practices that serve as learning goals or outcomes. However, implementing programming-based mathematics instruction requires teachers to have sufficient knowledge about learners' thinking processes in such a context, while very little is known about multifaceted solution development from a learner-centred perspective.

Objectives: Viewing CT and MT as processes that go beyond specific skills or concepts, we conducted a qualitative study to investigate how participants develop computational solutions to mathematical problems and construct meaningful understandings of these solutions.

Methods: We adopted an interpretive approach to participants' solution pathways to reveal their diverse thinking processes underlying solution development. A constant comparative analysis approach was undertaken to guide the data analysis.

Results and Conclusions: We identified multiple solution pathways in developing programming-based mathematical solutions (PMS) and characterised four significant pathways comprising seven distinct sub-situations: (1) transition between personal MT and invalid PMS, (2) evolution from invalid PMS to valid PMS, (3) construction from non-meaningful PMS to meaningful PMS and (4) revision from suboptimal PMS to optimal PMS. The findings contribute to a deeper understanding of problem solvers' learning in programming-based mathematical problem-solving and offer implications for theory and practice in programming-rich mathematics education.

1 | Introduction

Computational thinking (CT) is regarded as an essential analytical skill for the 21st century (Wing 2006). Given its position at the centre of disciplinary and interdisciplinary (e.g., STEM) practices, it has drawn increasing attention in K-12 educational research. On one hand, CT encompasses not only problem-solving competence but also algorithmic practices and data literacy, which are increasingly called upon in the scientific fields, including areas in AI and machine learning. On the other hand, mathematical thinking (MT) includes a flexible understanding of mathematical concepts, general reasoning skills and familiarity with heuristic strategies (Stacey 2006). As a 'style of thinking that is a function of particular operations, process and dynamics recognizably mathematical' (Burton 1984, 35), MT can be regarded as

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2025 The Author(s). *Journal of Computer Assisted Learning* published by John Wiley & Sons Ltd.

Summary

- What is currently known about this topic?
  - Research evidenced students' development of computational thinking (CT) and mathematics learning during programming-based mathematical problem-solving.
  - Research focus has been on specific CT concepts or practices as learning goals or outcomes.
  - Little is known about learners' thinking processes in developing programming solutions.
- What does this paper add?
  - Viewing CT and MT as problem-solving processes, we propose a framework that characterises the state of the solution development to offer insights into problem-solvers' programming processes.
  - Four significant pathways comprising seven distinct sub-situations in developing programming-based mathematical solutions (PMS) are characterised to interpret problem-solvers' thinking.
  - The reconstruction of a suboptimal PMS into an optimal PMS has the potential to deepen problem-solvers' understanding of the problem and advance their co-development of CT and MT.
- Implications for practice/or policy
  - Provides a way for educators to discern and understand students' learning and be responsive to students' thinking in a programming-based mathematics classroom.
  - Informs instructional approaches to enhance students' engagement and understanding of programming-based mathematical problem-solving.
  - Informs teacher professional development initiatives to focus on student thinking in programming-based mathematical problem-solving.

sharing common characteristics with CT when approaching a problem-solving situation (e.g., Ng et al. 2023; Sinclair and Patterson 2018). Related to CT, programming is deemed a fundamental practice in computer science and modern mathematical endeavours, which supports solving cognitively demanding tasks through abstraction, analysis and automation (Grover and Pea 2013; Tikva and Tambouris 2021). Thus, taking programming as a tool for problem-solving, programming-based mathematical problem-solving is an interdisciplinary approach that involves the integration of CT and MT to solve mathematical problems. This approach emphasises the benefits of integrating CT and MT to develop problem-solving capabilities, offering a dynamic and synergistic way to engage with computer science and mathematical concepts (e.g., Cui and Ng 2021; Kaufmann and Stenseth 2021; Miller 2019).

Different orientations of the interplay between CT and MT during programming-based mathematical problem-solving are identified in Ye, Liang, et al.'s (2023) systematic review, which include applying mathematical knowledge to anticipate and interpret computational outcomes or to mediate debugging practices, as well as generating new mathematical knowledge alongside CT development. However, most of the reviewed studies characterised CT as a set of specific skills or components (concepts and practices) related to computer science as a learning goal or product (Tang et al. 2020) rather than 'captur[ing] how CT develops and functions as a dynamic and iterative process of testing, debugging, and fixing mistakes to achieve a computational goal' (Kopcha and Ocak 2023, 1). This research gap leaves us with a lack of knowledge about how learners think at a fine-grained cognitive level, highlighting the need for in-depth investigation of thinking processes in programming-based mathematical problem-solving. As Grover et al. (2017) argued, '*how* one goes about developing programs is just as important as *what* one produces' (p. 2). Therefore, examining the solution pathways has significant implications for classroom teaching, as teachers require a theoretical framework with a process perspective to support their interpretation of and response to students' thinking.

To this end, we set out to investigate problem solvers' programming solution pathways to gain insights into their way of thinking during programming-based mathematical problem-solving, with the aim of providing empirical evidence to inform pedagogical approaches in K-12 programming-based mathematics instruction. Specifically, we adopt an interpretive approach to one's solution pathways to reveal the diverse thinking processes underlying solution development—from a learner-centred perspective, without judging their solution against the researcher's, thereby implying that there is no standard solution path. This is especially pertinent given the programming tool's affordance to perform procedures and express output in diverse ways and the open-ended nature of associated task designs (Repenning and Basawapatna 2021; Zhang and Nouri 2019). Viewing thinking processes that go beyond specific skills or concepts, in this study, we focus on how participants navigate their CT and MT in developing programming-based mathematical solutions to address the following research question: *What solution pathways do problem solvers develop while engaging in programming-based mathematical problem-solving?*

2 | Empirical and Theoretical Background

2.1 | CT and Programming

CT refers to the thought process involved in formulating problems and representing problems so that their solutions can be represented as computational steps and algorithms (Aho 2012) or in a form that can be effectively carried out by an information-processing agent (Wing 2011). In line with this conceptualisation, the componential definition of CT is constructed in a way that draws on computing concepts and practices (e.g., Brennan and Resnick 2012; Grover and Pea 2013; Mannila et al. 2014), such as variable, conditional and algorithmic thinking. Conceptualising CT in mathematical endeavours, Weintrop et al. (2016) proposed a taxonomy comprising data practices, modelling and simulation practices, computational problem solving practices and systems thinking practices. Further, in Ye, Liang, et al.'s (2023) systematic review, they offered an integrated interpretation of how different computational concepts and practices are operationalised and illustrated in CT-based mathematics contexts. For example, the term 'conditional' refers to 'a set of decisions,

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

actions, statements executed based on numerical, algebraic, or geometric properties' (p. 14) in a programming-based mathematical context.

While CT is often conceptualised as a componential construct, research on CT has predominantly approached it as a summative learning outcome. On one hand, researchers have employed quantitative methods to evaluate students' learning outcomes in CT after completing certain integrated mathematics and CT tasks (e.g., Dickes et al. 2020; Hu and Wang 2024; Rodríguez-Martínez et al. 2020). For example, Rodríguez-Martínez et al. (2020) utilised the Computational Thinking Test to assess students understanding of four CT concepts—sequences, iterations or loops, event handling and conditional statements—after they solved problems involving mathematical concepts using Scratch. On the other hand, researchers have adopted qualitative methods to gather evidence of CT development by analysing students' mathematical and programming performances (e.g., Mateus et al. 2024; Pei et al. 2018; Sinclair and Patterson 2018; Yeung et al. 2024). Overall, while many studies have taken a summative approach to assess students' learning outcomes in CT, the lack of a macro dynamic perspective and formative assessment may hinder teachers' capacity to promptly notice, understand and respond to students' thinking as they deal with programming-based problems.

2.2 | CT and Problem Solving

Alternatively, CT can be seen as a set of competences requiring students to develop both domain-specific knowledge and problem-solving skills (Kalelioglu et al. 2016; Selby and Woollard 2013; Tang et al. 2020). For example, Selby and Woollard (2013) defined CT as 'a focused approach to problem solving, incorporating thought processes that utilize abstraction, decomposition, algorithmic design, evaluation, and generalizations' (p. 5). In contrast to summative learning outcomes, when CT is perceived as a dynamic process of problem-solving, research on the integration of CT and mathematics emphasises the evolving nature of the problem-solving progression. Some studies have concentrated on the CT-based problem-solving process associated with specific mathematical engagement, for example, Aminah et al. (2022) examined the thinking process of prospective teachers in solving linear Diophantine equation problems, while others have investigated the general thinking process when learners engage in CT-based mathematical activities, such as Lee et al.'s (2011) introduction of the Use-Modify-Create three-stage progression framework to represent the phases of students' cognitive and practical activity. Regarding the context of programming-based mathematical problem-solving, Kaufmann and Stenseth (2021) developed a model (Figure 1a) to characterise how mathematics mediates the complex process of debugging when students apply programming to solve a mathematical problem. Figure 1a illustrates how mathematics mediates the debugging process through the iterative processes of testing, observing, analysing, forming hypotheses and changing programs. Furthermore, by analysing the orientation, planning, execution and checking phases of participants' problem-solving, Hu et al. (2025) proposed a data-centric problem-solving framework that incorporates problem-solving actions, resources and methods and heuristics, highlighting the interplay between data exploration and analysis as well as the actions of recasting data and creating derived objects (Figure 1b).

Taken together, these studies collectively underscore CT as an iterative, dynamic process in facilitating mathematics learning. They examined various dimensions of the thinking or developmental processes involved in specific mathematical topics (e.g., Aminah et al. 2022), particular problem-solving domain (e.g., Hu et al. 2025) and general problem-solving processes integrating CT and mathematics (e.g., Kaufmann and Stenseth 2021; Lee et al. 2011). While these studies offer valuable insights into how learners develop specific concepts or engage in CT activities, a gap remains in our understanding of the problem-solving process learners undergo when

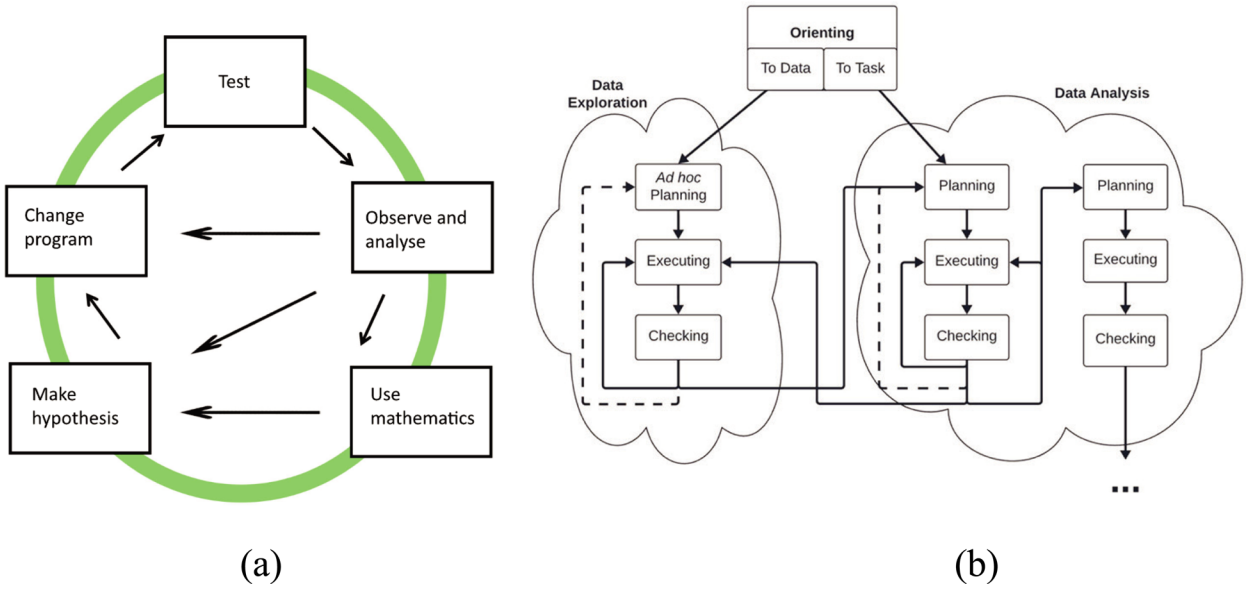


FIGURE 1 | (a) Iterative process (Kaufmann and Stenseth 2021); (b) initial framework for CT when engaging with data (Hu et al. 2025).

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

they develop solutions to programming-based mathematical problems. Identifying learners' solution pathways in this context plays a significant role in supporting teachers' classroom practices, as such insights empower teachers to better notice, elicit, interpret and respond to students thinking—an essential aspect of subject-specific teaching and a critical dimension of teacher professional development programs (Kyriakides et al. 2024; Shure et al. 2025; Zbiek et al. 2024). Although the aforementioned models summarise the overall process of programming-based problem-solving, they does not fully characterise how the involved processes are mutually transformative from both CT and MT perspectives. Furthermore, as in Kaufmann and Stenseth's (2021) and Hu et al.'s (2025) framework, the nodes of the flowchart represent actions and practices at a behavioural level; there is a lack of in-depth understanding regarding the stages of learners' solution generation and the transitions between these stages. Consequently, there is a need for a theoretical framework that goes beyond the development of specific kinds of CT or MT but explicitly connects the roles of CT and MT during the solution development process, enabling researchers and educators to recognise and make sense of one's solution in programming-based mathematical problem-solving, thereby supporting K-12 programming-based mathematics instruction.

2.3 | The Current Study

Before the concept of CT (as proposed by Wing 2006) became prevalent, Papert (1980) has emphasised the value of computers as 'objects-to-think-with' to engage students in disciplinary learning. He envisaged a paradigm shift in school mathematics education, wherein CT facilitates the constructionist practice of concretising abstract mathematical concepts and building personal mental models to understand the world. Echoing the integration of CT in mathematics education, in this study, we adopt a hybrid perspective that combines Wing's and Papert's conceptualisations of CT. Specifically, we view CT not only as a way of thinking about problem solving in a programming environment but also as a tool for learning disciplinary concepts in mathematics and engaging in mathematical practices. In doing so, we propose a perspective to characterise a solution process in which the nodes of the flowchart represent the evolving states of solutions to programming-based mathematical problems.

According to Jonassen (1997), problems are defined by the problem domain, problem type, problem-solving process and solution. Specifically, the concepts, rules and principles that characterise the problem elements determine the problem domain. Contextualising Jonassen's (1997) notion of the problem domain in the current study, we define a *programming-based mathematical problem* as one that necessitates the coordination of MT and CT, specifically that of mathematical concepts and programming rules, to form a solution strategy within a programming environment. A *programming-based mathematical solution* (PMS) is thus conceptualised as a set of programming codes developed along with associated algorithms and strategies, from which the co-development of MT and CT evolves. To gain insights into how problem solvers develop their PMS, we analyse their *solution pathways*, which refer to the specific route a solver takes to form a solution between or within different problem-solving stages, ultimately facilitating the formation of a PMS. In this study, we detail participants' solution pathways and illustrate how they orchestrate CT and MT while using different computational strategies and practices for constructing, developing and evaluating their PMS.

3 | Method

This study adopted a qualitative interpretative research design, aiming to construct meanings, narratives and (mental) images by understanding how things unfold (Wiesner 2022). Interpretative research employs an inductive approach that grounds with data; it is appropriate for studying context-specific, unique, or idiosyncratic events or processes and is also helpful for theory construction in areas with no or insufficient priori theory (Bhattacherjee 2012). As the purpose of this study was to explore how participants develop solutions to programming-based mathematical problems, it required understanding not only 'what' happened but also interpreting 'how' it happened (Wiesner 2022), in the sense of problem solvers' cognitive processes while engaging in programming-based mathematical problem-solving. Adopting an interpretative analysis of participants' discourse and programming solutions allows us to recollect the events through the participants' vision or thinking pattern to construct meaningful explanations of their problem-solving process.

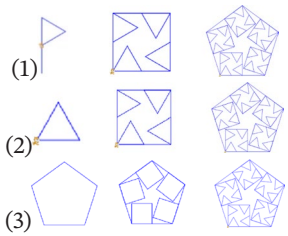
3.1 | Participants and Context

This study is part of a larger design-based research project which comprised a series of classroom teaching experiments and teacher professional development workshops, with the aim to (a) develop a series of tasks to support the practice of computationally enhanced mathematical problem-solving in school mathematics classrooms and (b) investigate students' and teachers' CT and MT development, problem-solving competence, as well as learning trajectories while engaging in the designed tasks.

The teacher workshops understudied were conducted during the second iteration of the design-based research cycle; they were designed to facilitate the professional development of school teachers in programming-based mathematics teaching and learning, using a series of refined tasks from the first cycle of classroom teaching. Six in-service mathematics teachers (four females and two males) from Mainland China volunteered to participate in this 10-h online professional development workshop in Scratch programming. The participants were chosen to represent a diverse range of teaching experience and had no prior knowledge of Scratch programming. During the teacher workshop sessions, they were divided into three pairs and attended the workshops in pairs. Over a 5-week period (2-h sessions weekly), the workshops began with familiarising the participants with basic Scratch programming and then gradually presented several programming-based mathematical tasks related to four mathematical domains (i.e., number, algebra, geometry and data handling). A think-aloud protocol was used to prompt the participants to explain their thinking to help us interpret their task solutions.

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 1 | Details of geometry and number tasks.

Topic domain	Task description	Possible solution strategies
Geometry (Session 3)	*Nested Figure Drawing*: Draw the figure shown below in Scratch. 	
Number (Session 4)	*Number Game*: Design a Scratch program that allows players to take turns inputting numbers and determining the outcome at the end of the game, which is to be played with the following rules: Two players take turns reporting numbers counting from 1. When they encounter a number that meets any of the following conditions, they need to skip it; otherwise, they lose the game: 1. Multiples of 7 2. Numbers that contain digit 7 in their decimal representation 3. Multiples of a number that contains digit 7 in decimal representation	*Possible Solution Strategies for Sub-Problem (3)*: Method 1: Enter *N*. Find all the numbers less than *N* that contain digit 7 in their decimal representation, and then test whether they are factors of *N*. If so, the player will lose the game. Method 2: Enter *N*. Find all the factors of *N*, and then test whether any factors contain the digit 7 in their decimal representation. If so, the player loses the game.

3.2 | Research Procedure and Task Design

To gain insight into participants' problem-solving process, we tasked participants to engage in a series of problems having no immediate known solutions. We also designed problems with multiple entry points, meaning that they can be solved in multiple ways and the problem solvers could choose any one way as their entry to the problem (Mason et al. 1985). We chose two representative sessions from the workshops to elaborate on the features of the tasks. The details of the two tasks are shown in Table 1.

In summary, both tasks have multiple entry points and multiple solutions, allowing us to gain insights into the participants' diverse solution strategies and observe their PMS development processes. Moreover, the two tasks address different mathematical domains and different types of CT competence (e.g., a number task focusing on algorithms vs. a geometry task focusing on pattern recognition). This allows us to examine the diverse interplays between CT and MT in different contexts. Taken together, the two tasks exhibit the potential to investigate the rich and diverse processes of generating PMS in response to the RQ.

3.3 | Data Collection, Processing and Analysis

As the participants engaged in the tasks, we encouraged them to think aloud and clarify their thinking to uncover participants' problem-solving processes. This includes when they encountered difficulties, made significant progress, or generated insightful interpretations of the problem. Each session was video-recorded and screen captured. The participants' programming artefacts and the researcher's fieldnotes were collected.

Our approach to responding to the RQ involved multiple stages of analysis. The first was to identify the important elements and phases that emerged during the problem-solving process, followed by generalising participants' programming solution pathways thematically. We reviewed and transcribed the video data, and then interpreted the data in terms of what the participants did and why they did that. Informed by the phases of problem solving formulated by Polya (1971), we first conducted deductive coding to categorise the participants' problem-solving process into four phases. Then, through constant comparative analysis (Strauss and Corbin 1990), we compared the episodes to generalise themes and defined key terms to represent the solution status, for example, valid and invalid PMS, (non-)meaningful and (sub-)optimal PMS. Finally, we reviewed the data and definitions again to check the generated typology of solutions to programming-based mathematical problems, as shown in Figure 2.

In Phase 1, understanding the problem mathematically means that a problem solver identifies the mathematical relationships underlying the problem from a mathematical perspective, which requires the application of their personal MT. In Phase 2, the problem solver devises a programming plan by analysing the task with respect to formulating the problem mathematically and computationally, as well as selecting appropriate strategies according to their available MT, CT and programming skills. They may also consider why certain programming approaches with mathematical considerations may facilitate a better solution. The outcome of this phase constitutes a PMS. In Phase 3, the problem solver follows the devised plan to build codes and monitors whether the plan is proceeding properly. Based on the programming feedback, they determine whether a PMS is *valid* or *invalid*. A valid PMS is one in which the problem solver expresses confidence in resolving the problem. When a problem solver runs the codes and considers

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

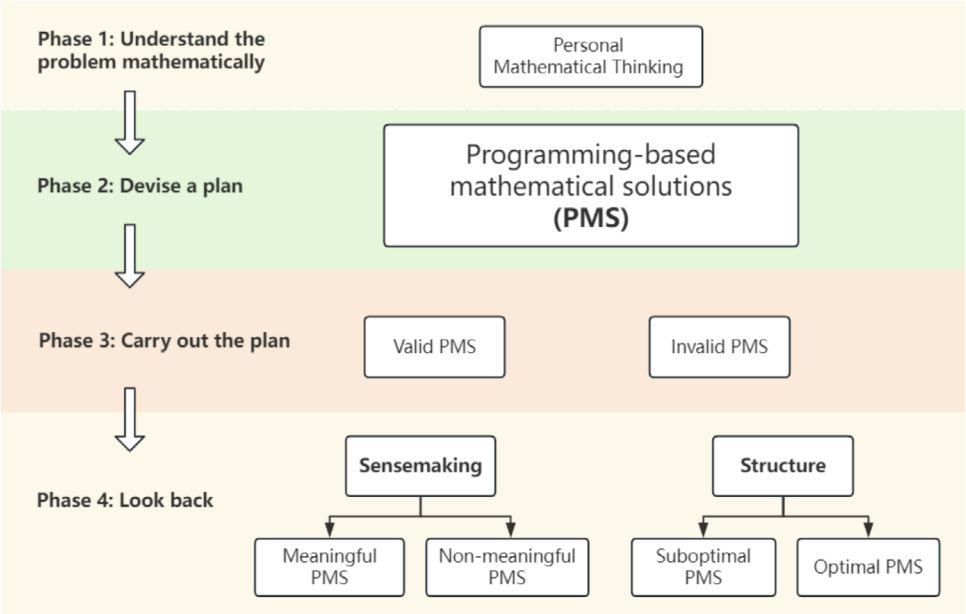


FIGURE 2 | Different types of solutions to programming-based mathematical problems.

them unsuccessful in solving the problem, this is considered an invalid PMS. Finally, we identify whether the problem solver can make sense of the solution and its structure when reviewing the PMS in Phase 4. A PMS is considered *non-meaningful* if the problem solver expresses confusion about the codes and the respective execution processes. In contrast, a *meaningful* PMS is one in which the solver clearly makes sense of the meaning of the program codes and their execution. In terms of solution structure, we consider a PMS to be *suboptimal* if the solver realises that the existing valid PMS can be improved toward a more systematic, logical and well-organised solution. Conversely, an *optimal* PMS is one with which the problem solver is satisfied and can be used as the final solution without any changes.

Upon identifying the typology of solutions, we thematically analysed the participants' programming solution paths. First, we adopted constant comparative analysis (Strauss and Corbin 1990) to iteratively compare and group themes, which involved identifying how the participants transitioned from one phase to another and from one particular solution status to another. Simultaneously, drawing upon prior work on programming-based mathematical problem-solving, we conducted deductive coding to identify practices facilitating the transition between different solution statuses, resulting in the following computational practices:

- *Translating and Restructuring*: Translating relates to what Ye, Ng, and Cui (2023) call translation flexibility, that is, choosing an appropriate strategy to translate between mathematical language (expression) and programming language (solution) in a given programming environment. Restructuring relates to integrated flexibility (Ye, Ng, and Cui 2023), which entails transformations between different programming solutions to the same problem by incorporating CT and MT strategies.
- *Incremental programming*: This means approaching a programming solution in small steps. It involves iterative cycles of imagining, changing and building particular elements of code, developing a little bit, trying it out and then developing further. This incremental process can be based on execution feedback, the programmer's experience and emergent ideas generated throughout the programming process (Brennan and Resnick 2012).
- *Code modelling*: This refers to coders interpreting the mathematics underlying a sequence of codes (Ye, Liang, et al. 2023), among which unplugged code modelling simulates each step of code execution in one's mind or in a paper-and-pencil context, while plugged code modelling refers to the behaviour that occurs in the programming environment, such as inserting a [wait X seconds] block to slow down the programming execution process to make it easier for participants to follow and navigate.
- *Systematic debugging*: There are various debugging strategies, for example, iterative refinement by trial and error, observing step-by-step execution or support from knowledgeable others (Brennan and Resnick 2012; Rich et al. 2019). Systematic debugging refers to debugging practices that take into account mathematical or programming considerations to identify, isolate, reproduce and ultimately correct unexpected problems in an efficient, structured manner.

Team discussions were held to examine the initial codes, themes and relationships between the solution pathways and underlying practices. Finally, by iteratively comparing and reviewing the data and themes, four solution pathways comprising seven sub-situations were eventually identified. We elaborate upon these findings in the following section.

4 | Results

Our analyses revealed four solution pathways: (a) transition between personal MT and invalid PMS, (b) evolution from invalid PMS to valid PMS, (c) construction from non-meaningful PMS to meaningful PMS and (d) revision from suboptimal PMS to

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

optimal PMS. We detail the specific solution pathways with representative examples.

4.1 | Pathway 1: Transition Between Personal MT and Invalid PMS

When confronted with a programming-based mathematical problem, problem solvers may draw upon their personal mathematical knowledge to form initial conceptualisations and strategies. In turn, when encountering an invalid solution, they may revisit their mathematical knowledge to reconstruct their strategies. Figure 3 characterises this bi-directional transition between invalid PMS and personal MT (Figure 3).

4.1.1 | Translating From Personal MT to Invalid PMS

Converting personal MT to an invalid PMS means that when an individual attempts to translate their MT into a programming solution, an error occurs during the translation process, which is likely due to mismatches between the mathematical and programming languages (Cui and Ng 2021). For example, in the Nested Figure Drawing task, Teacher Z first proposed drawing a 'flag' as a base figure with multiple rotations to form a 'square with triangles' (see the first possible solution in Table 1). However, after completing the codes to draw a 'flag', he found it challenging to centre-rotate the base figure. The following discussion unfolded between the researcher and Teacher Z about his struggle to translate his personal MT into the programming environment:

1. **R (Researcher)**: How do you think it was drawn?
2. **Z**: It looks like a flag, then rotated around a centre in the middle ...

[Z tried to draw a flag standing vertically. After about 8 min, he drew the flag.]

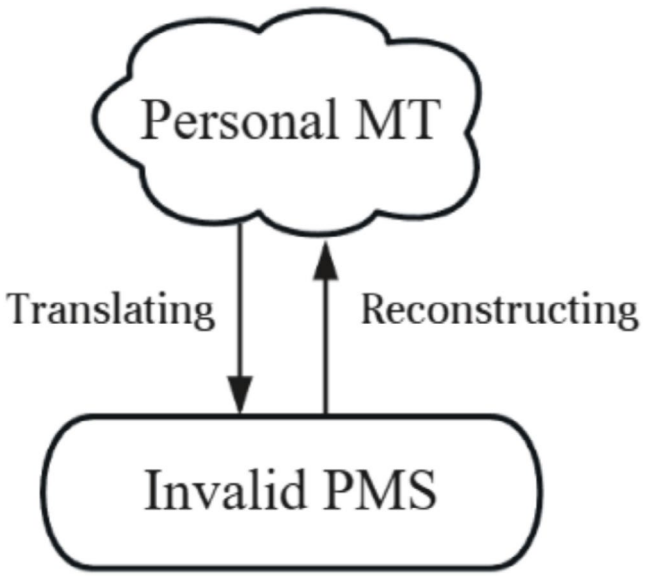


FIGURE 3 | Pathway of transitioning between personal MT and invalid PMS.

3. **R**: Then, how would you rotate it?
4. **Z**: Is it possible to rotate it as a component? I want to use this flag as a base figure, and then rotate this base figure to the right ... See it as a whole and rotate it ... rotate it four times, no, just three times, one, two, three, four, it should be (drawn) ... How can I do that? Create a new (subroutine) block first, isn't it?
5. **R**: Well, sure. You can put it in a (subroutine) block, and then when you want to use it, you can just call it up ...
6. **Z**: Yes, I'll give it a try. Build a block ...The name of the block is the base unit. And then put it ... and then if I have rotated it ...

[Thinking about how to rotate, about 1 min later]

7. **R**: Can you find the centre of rotation?
8. **Z**: It seems like ... yes, we have to find the centre of the rotation. Is it possible to find the centre of the rotation? Is this achievable?

We interpret that Teacher Z had a strategy for drawing the target figure in that he was able to analyse and decompose the geometric figure and conceive the target shape as a result of transforming the base figure [Turn 4]. That is, his *personal MT* was available. However, he struggled with the process of rotating the base figure in the programming environment. He even doubted his approach because he could not find a feasible way to centre the rotation in a programming context [Turn 8]. We suggest that this meant he had trouble *translating* the process of drawing the figure in a programming environment, in which the problem solver must command the programming tool to complete the drawing with explicit instructions and executable codes that the programming tool 'understands'. Experiencing difficulty with the *translation* resulted in his *personal MT* being transformed into an *invalid PMS*. We characterise this situation as a problem-solving episode in which the problem solver did not translate his personal MT into the corresponding programming solution.

4.1.2 | Reconstructing From Invalid PMS to Personal MT

An invalid PMS requires debugging (Kaufmann and Stenseth 2021), while our analysis suggests that instead of debugging by wrestling with the buggy computer program itself, some participants decided to *reconstruct* their MT to generate a new mathematical strategy to tackle an *invalid PMS*. For example, when working on the Number Game, Teachers Q and T each proposed their own solution to the third condition (i.e., multiples of a number that contains digit 7 in the decimal representation cannot be reported). Teacher Q first decided to adopt his proposed mathematics strategy (i.e., dividing N by 7, 17, 27, 37... to check if the remainder is 0) and translated it into the corresponding PMS. Teacher T, on the other hand, had questioned Teacher Q's solution but was unable to specify what the problem was. In this case, the researcher offered a 'special number' that did not fulfil Teacher Q's solution to facilitate their realisation that it was an invalid PMS:

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

9. **R**: What about 71? 142 is twice as much as 71. Then, you cannot examine it *[interrupted by Teacher Q]*.
10. **Q**: 71, oh, I understand, you mean that 71 is not within the range of a number by adding 10, and it is a multiple of a number containing 7.
11. **R**: Yes, or you can add a condition to check this kind of number.
12. **Q**: Let's exclude it. The multiples of number containing the digit 7, such as 71 and 72, the multiple (of 71) is 142. Then, we can first determine whether the number containing the digit 7 is contained in this condition.
13. **T**: Multiples of 701 do not contain the number 7. Therefore, we have to consider the numbers starting with 7, such as 771. There is no such thing as 771 multiplied by 2. I think this seems difficult to do.
14. **R**: Let's limit it to 100 first, if that doesn't work.
15. **Q**: There will be contradictions within 100, too, so I want to think about how to solve it. 71...72 ... *[Half a minute later]* In fact, the solution (Teacher T) just proposed is also feasible. That is, we keep dividing a larger number and determining whether it is a number containing 7.

As the transcript shows, Teacher Q engaged in reflection after noticing the existence of special cases and the MT underlying them. Despite the researcher's suggestion that the conditions could be built upon to develop a valid PMS [Turn 11], Teacher Q gave up on debugging the existing *invalid PMS* and decided to develop a new PMS based on Teacher T's MT [Turn 15]. Hence, we characterise this episode as illustrating a situation in which the underlying MT is unproductive in generating a valid solution, and subsequent discussion and reflection on their MT facilitate problem solvers to reformulate the problem situation and solution strategy. This pathway is featured by the problem solvers' *reconstruction* or modification of the underlying algorithm and mathematical ideas to improve their *invalid PMS*. The next section discusses our analyses of the evolution from invalid PMS to valid PMS.

4.2 | Pathway 2: Evolution From Invalid PMS to Valid PMS

We observed numerous occasions in which the participants were able to revise their solutions to *evolve* their *invalid PMS* to *valid PMS* through different pathways (Figure 4). We elaborate on these pathways below.

4.2.1 | Systematic Debugging From Invalid PMS to Meaningful PMS

When evolving an invalid PMS into a valid PMS, one possible avenue is *systematic debugging*, which leads to a meaningful valid PMS. As mentioned in Section 4.1.2, Teachers Q and T adopted a new mathematical strategy to reconstruct a program (i.e., dividing N by an increasing number (1, 2, 3, ..., N) to see if the divisor contained digit 7 in the decimal representation). However, by inputting numbers to test their programs, they found that the new PMS was still invalid. They then worked on debugging the *invalid PMS*

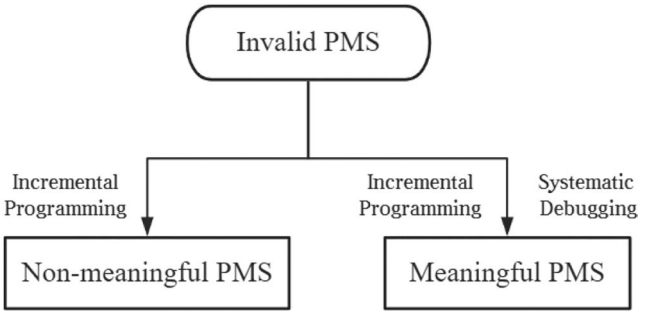


FIGURE 4 | Different pathways of evolution from an invalid PMS to a valid PMS. The rounded rectangular box denotes an invalid PMS, and the rectangular boxes denote a valid PMS.

by engaging in self-reflection and realised that 'checking whether the remainder contains digit 7' was the incorrect way to proceed:

16. **Q**: What we are judging now is the remainder (Figure 5a), but what we want to judge is [...] the divisor, not the remainder ... So, it's not that the remainder contains 7. It wouldn't work when there was a remainder, so it must be divisible.
17. **T**: But why did our previous solution seem to work?
18. **Q**: Because the numbers we just entered are all greater than 7... For example, 17 divided by 10 equals 1, and the remainder is 7, isn't it? ... (What we should check is) not the remainder but the divisor. It must be divisible, which means that we should use if-then-else.
19. **T**: If the remainder is 0, then the quotient contains 7.
20. **Q**: Yes, the remainder must be 0, so we check whether the quotient contains 7... Okay, I probably understand; let's think about it. This thing (the remainder) is equal to 0. Then we make another judgement ... The number input containing 7 will not work. So, it doesn't work if the divisor contains 7 or the quotient contains 7... Do we have to make three judgements? First, enter a number that does not contain 7. After it has been judged, the divisor and quotient cannot contain 7 either, right?
21. **T**: The divisor cannot contain digit 7?
22. **R**: Why not?
23. **T**: Yes, why?
24. **Q**: Well, the divisor can contain 7, but that ...
25. **T**: The quotient can't.
26. **Q**: Yes, the quotient cannot contain 7.

The above episode shows that Teacher Q attended to the buggy code in the test condition, that is, it was erroneous to test whether the remainder contained digit 7 or not. The test to be undertaken is to check whether the quotient contains digit 7 in the case where the remainder is 0 (Figure 5b). In this debugging process, the two teachers reflected on the test condition with a specific purpose [20]. Specifically, they reflected on the object of the test—namely, the remainder, divisor and quotient—and the conditions of the test (whether the quotient contains digit 7 when the divisor is divisible). Because such debugging behaviour was well planned and purposeful, without trial and

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

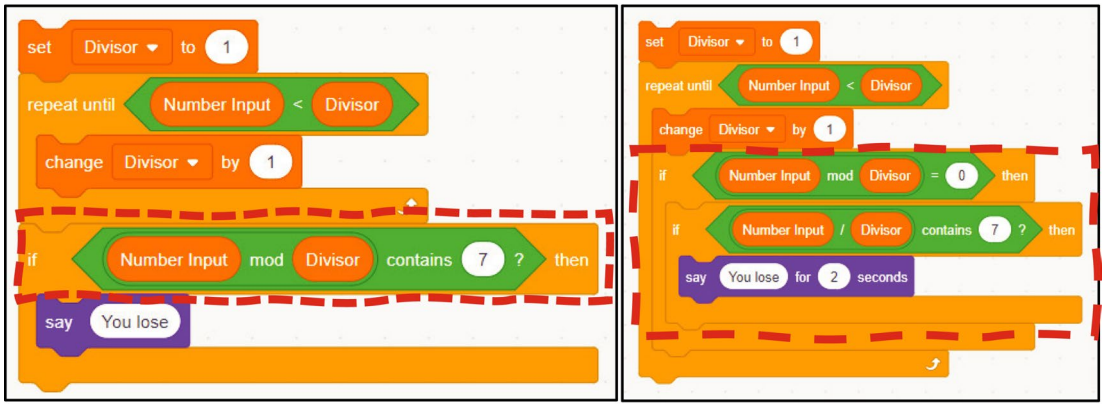


FIGURE 5 | (a) Invalid PMS due to the mistranslation of the test condition; (b) valid PMS developed by systematic debugging of invalid PMS in (a).

error, we considered it to be evidence of *systematic debugging*. Moreover, the two teachers reached agreement on the doubts that arose during the debugging process and on the approach to debugging. This suggests that their understanding of the underlying MT was clear and unambiguous and that the new PMS was not only *valid* but also *meaningful* to them.

4.2.2 | Incremental Programming From Invalid PMS to Meaningful PMS

While *systematic debugging* refers to the process of debugging with mathematical and programming consideration in a purposeful manner to obtain a valid PMS, *incremental programming* describes the progressive programming behaviour toward the construction of a PMS. For example, Teacher Z's *incremental programming* involved building on the base shape to produce the target nested figure. As shown in Table 2, he first followed the codes for drawing the horizontal 'flag' (see Step 1) to draw the second flag (Step 2), followed by drawing the third flag and one triangle (Step 3). Noticing that he was missing one side of the square, Teacher Z finally appended the [move 100 steps] and [turn left 90°] blocks to generate the fourth side of the square and to orientate the Pen to face forward, respectively (Step 4). Here, Teacher Z's *incremental programming* is demonstrated by his completing the codes step-by-step to visualise his programming output at each step. While both yielded a *valid PMS* from an *invalid* one, Teacher Z's pathway was different from Teacher Q's *systematic debugging* in that he frequently executed the codes to obtain positive, stepwise feedback from the program before writing the next step, whereas Teacher Q finished all the codes and then ran the program to see if it was workable.

We interpreted that Teacher Z's PMS was *valid* as well as *meaningful* because he demonstrated the clear goal he wanted to achieve before each attempt (e.g., he uttered 'place a triangle and then move it 100 steps') and also considered the programming output as in line with his expectation. Even after the third step, when he noticed that one side was missing and the direction of the Pen was downward, he quickly identified what to do next. Furthermore, after the final shape was drawn, he swiftly restructured the codes with the 'repeat' block. Thus, we interpreted that this PMS was meaningful to him, since he was well informed about each step of the respective PMS.

4.2.3 | Incremental Programming From Invalid PMS to Non-Meaningful PMS

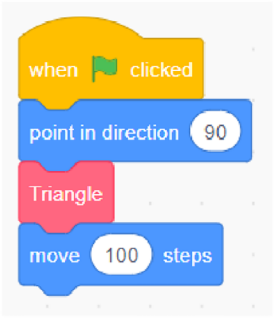
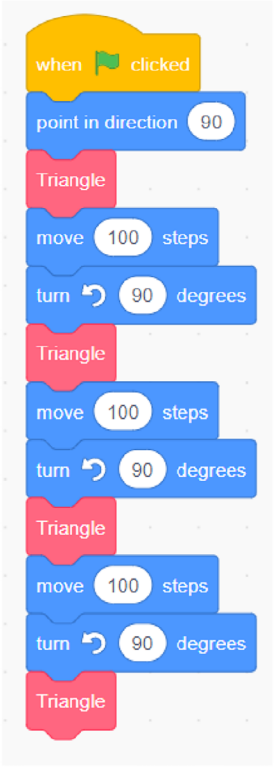
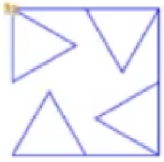
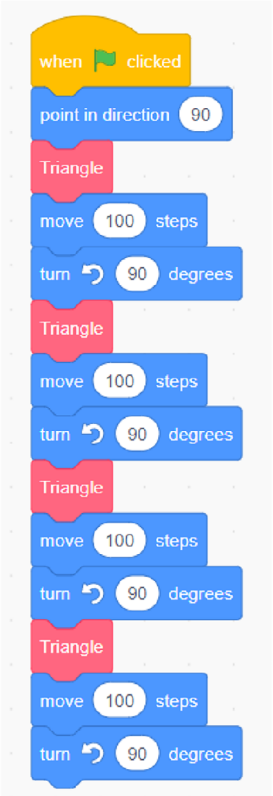
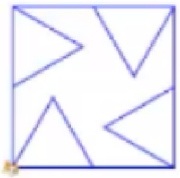
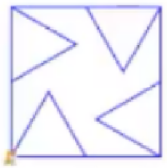
We observed that not all incremental programming results in a *meaningful PMS*; occasionally, our participants did not make sense of the program, even though their solution was *valid*; hence, we call this a *non-meaningful PMS*. When engaging in the Nested Figure Drawing task, Teacher Y proposed the mathematical strategy of drawing a pentagon first, then multiple squares within the pentagon, and finally triangles within each square (see the third possible solution in Table 1). When implementing her second part of the plan to draw five squares within a pentagon, Teacher Y developed the following program in small steps: first, she remixed a subroutine to draw the first square on one of the sides of the pentagon (Figure 6a); then, she changed the position of the Pen (Figure 6b) to draw the second square (Figure 6c); and finally, she used the loop instruction to make a pentagon with five nested squares (Figure 6d).

We observed that Teacher Y developed this valid PMS by incremental programming, meaning that she coded one step first, then tested it out, and then developed it further based on the execution feedback. However, despite the result of the programming fulfilling the task requirements, Teacher Y was clearly confused about the process:

27. **R**: The figure (a pentagon with nested squares) is drawn. Can you see how many times the square needs to be repeated? What do you think?
28. **Y**: Five times is enough. [Teacher Y removed the second 'repeat' module in Figure 6d]. But I'm a bit confused.
29. **R**: What's wrong?
30. **Y**: Because I've set the Pen to rotate every time it finishes drawing one side and to continue to draw (the other sides). Then, it will move 100 steps after the (square) is done. But why is it like this here? *[referring to the Pen*

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 2 | Teacher Z's incremental programming process to generate a meaningful PMS.

Codes	Programming Artifact	Codes	Programming Artifact
Step 1		**Step 2**	
			
Step 3		**Step 4**	
			 Generating the fourth side of the square  Making the Cat face forward

position and orientation being the same as Figure 6a] I'm just confused.

Teacher Y expressed her confusion about the drawing process and her uncertainty about the position of the Pen after drawing each square [Turn 30]. This suggests that the *valid PMS* did not appear to be meaningful to Teacher Y. In the next section, we elucidate another pathway, showing how it is essential for the problem solver to comprehend a non-meaningful *PMS* through *code modelling*.

4.3 | Pathway 3: Construction From Non-Meaningful PMS to Meaningful PMS

As shown previously, a problem solver may fail to make sense of a valid PMS. Here, we evidenced that *code modelling* is one way to facilitate the construction of a *meaningful PMS* from a *non-meaningful* one (Figure 7).

Recall that *code modelling* is an individual's step-by-step simulation of the behaviours of a sequence of codes to understand

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

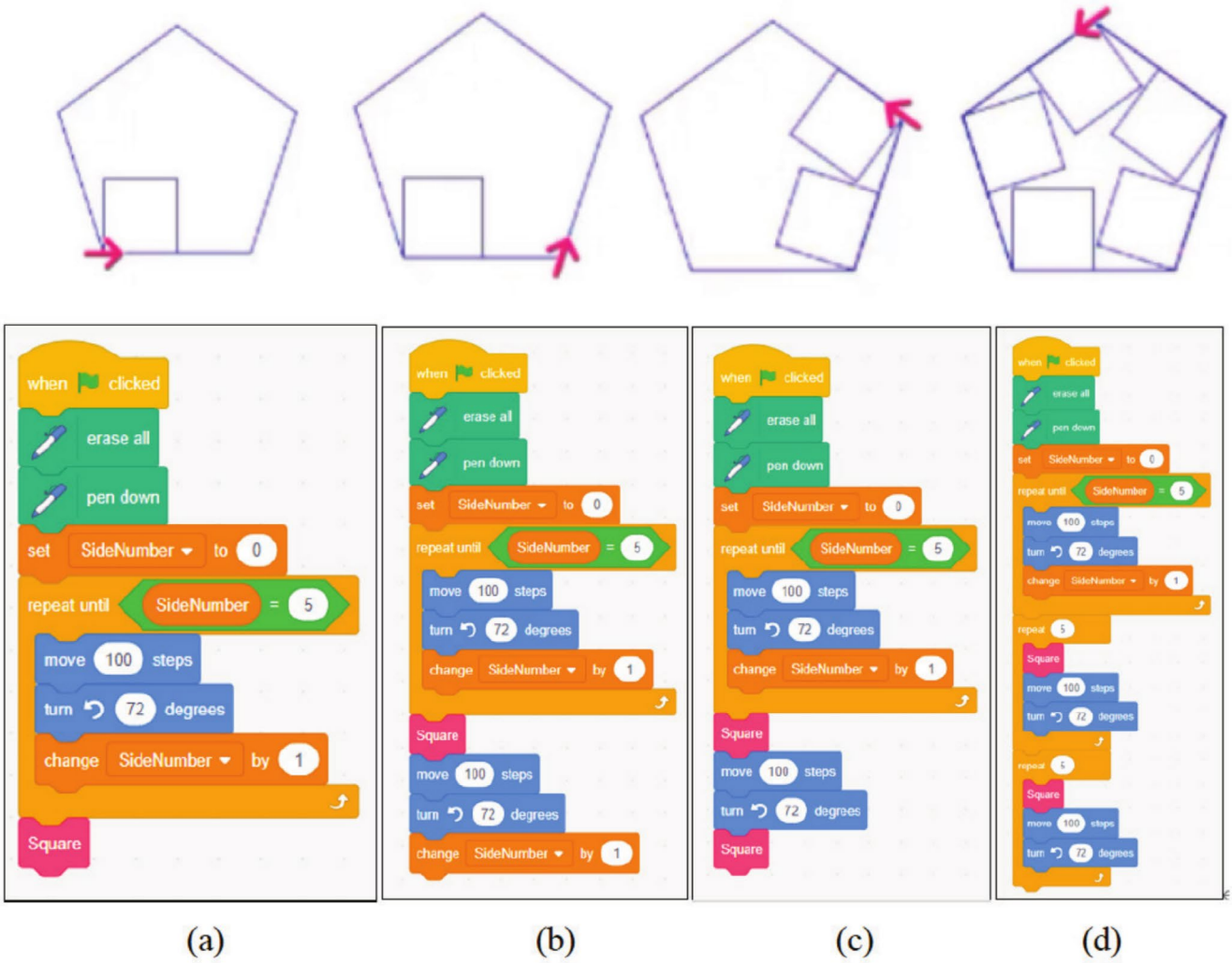


FIGURE 6 | Teacher Y's incremental programming: (a) draw the first square on one of the sides of the pentagon, (b) change the position of the PEN, (c) draw the second square in the pentagon and (d) use the loop instruction to make a pentagon with five nested squares.

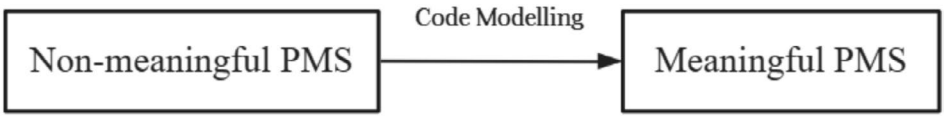


FIGURE 7 | Code modelling from a non-meaningful PMS to a meaningful PMS.

how they work or why they do not work (Ye, Liang, et al. 2023). As delineated in Section 4.2.3, Teacher Y generated a *valid* yet *non-meaningful PMS* through *incremental programming*, but she was confused about how the program worked to produce the diagram. The episode below describes how, through *code modelling*, she was able to resolve her confusion:

31. **Y**: Because I've set the Pen to rotate every time it finishes drawing one side and to continue to draw (the other sides). Then, it will move 100 steps after the (square) is done. But why is it like this here? I'm just confused.
32. **R**: What do you mean, 100 steps is not right?
33. **Y**: After drawing a square, it should take 50 steps...Oh, wait a minute. Well, after drawing (the square), the Pen returns to the starting point! Back to the starting point after drawing, right? Then, there's no problem now.

Here, Teacher Y simulated each step in her utterance by breaking down each line of code with the help of the generated figures [Turn 31], thus successfully *constructing the meanings* of the PMS. That is, she could finally make sense of the Pen position and clarify her new understanding of the code [move 100 steps]: the Pen would go back to the starting point when the drawing of five squares was completed (see Figure 6a) [Turn 33]. Teacher Y's *code modelling* not only required her to apply her CT skills to make sense of the code execution sequence but also to make a connection with her MT to interpret the programming output and the process it constituted.

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

4.4 | Pathway 4: Revision From Suboptimal PMS to Optimal PMS

The last pathway we observed is the *revision* from a *suboptimal PMS* to an *optimal PMS* after a *meaningful PMS* is constructed. That is, the problem solver realises the possibility of optimising the PMS. As will be demonstrated in this pathway, they orchestrated their CT and MT to restructure the codes of the suboptimal PMS to make it more concise and efficient (Figure 8).

The process of *restructuring* a *suboptimal PMS* to an optimal *PMS* implies the problem solver's increased ability to coordinate their MT and CT. To illustrate, we discuss Teacher Z's and Y's restructuring of an existing valid PMS in the Nested Figure Drawing task. In the case of Teacher Z, he proposed the idea of improving the PMS when the researcher posed the question of whether there was some sort of pattern in the codes.

34. **R**: Do you notice any pattern (in this PMS; see Figure 9a)?
35. **Z**: ... The (blocks) here (in the main program) are all the same.
36. **R**: Yes, and what determines them?
37. **Z**: It's determined by how many sides it has ... If it's a pentagon, it's going to have five blocks, then four for a quadrilateral, and three for a triangle.

[About half a minute later]

38. **Z**: Can these blocks be more concise?
39. **R**: Yes, what do you think?
40. **Z**: How about a variable? No, let me think about it. When repeated, there should be ... something like an event, right? *[Looking for the appropriate block in the toolbar]* Loop? Executing the codes repeatedly? Oh, yes, I remember it. *[Teacher Z reconstructed the code using a loop block.]* Repeat five times? Yes, that's right. *[Teacher Z continues to simplify the other two subroutines by using a loop block; see* Figure 9b*].*

Teacher Z's programming process of *restructuring* involved optimising the code structure through his integration of MT and

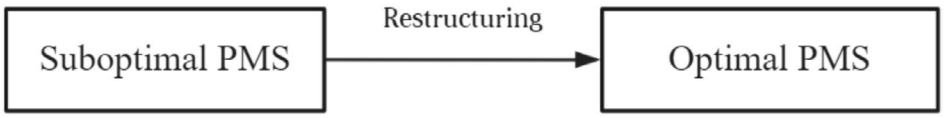


FIGURE 8 | Restructuring from a suboptimal PMS to an optimal PMS.

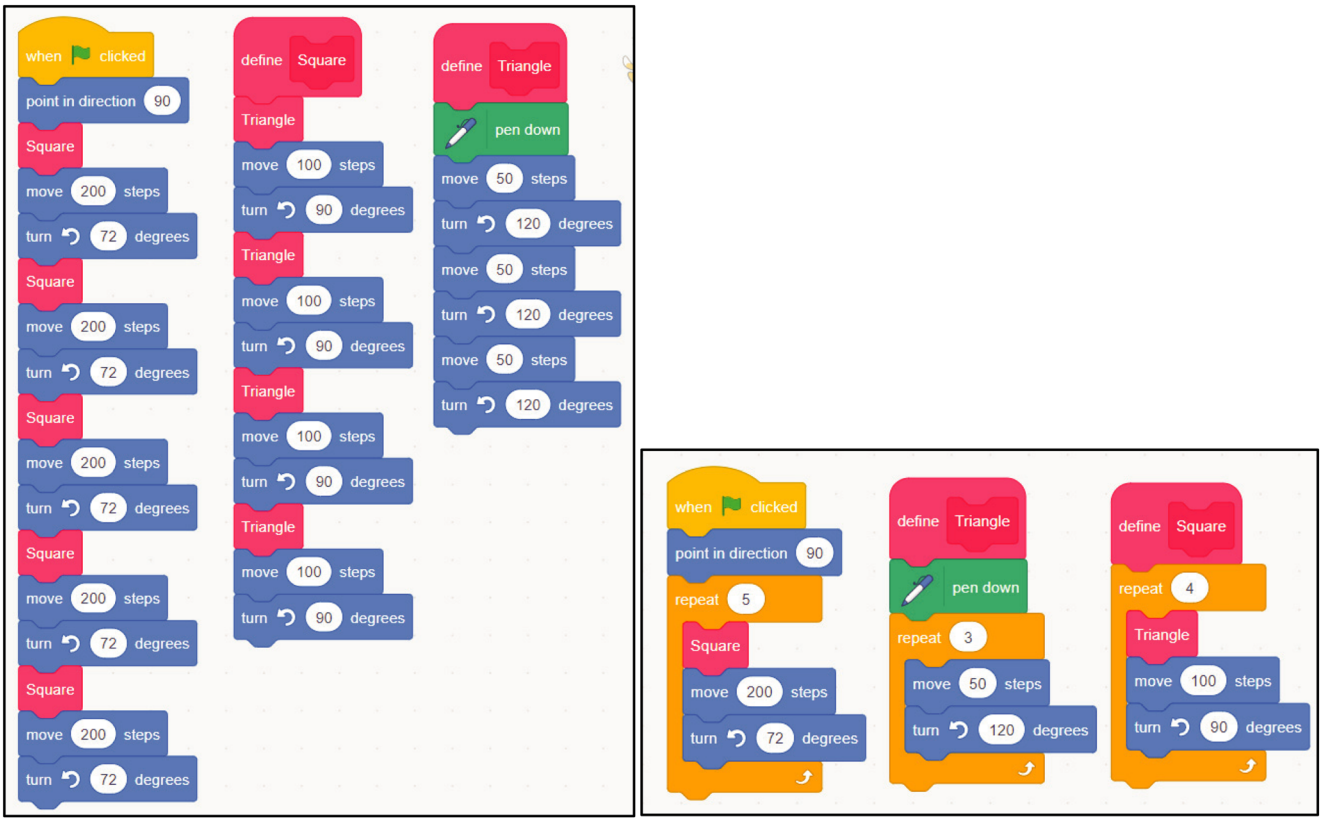


FIGURE 9 | (a) Teacher Z's original PMS before restructuring; (b) the optimal PMS with the use of a loop.

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

CT. Although Teacher Z's use of the loop to simplify the codes seems to be exclusively relevant to CT, in which the loop is an important CT concept (Brennan and Resnick 2012), this idea is more than purely CT. Analysing the transcript in detail, we note that the researcher posed a programming-related question about observing the code patterns [Turns 34 and 36], to which Teacher Z responded in a mathematical sense: he recognised that the number of times the code repeated was related to the number of sides of the regular polygon [Turn 37].

In contrast, Teacher Y optimised the process of PMS differently from Teacher Z. She first reflected on the drawing process of the *suboptimal PMS* and then activated her CT knowledge (e.g., decomposition, loop and subroutine) to *restructure* the codes toward developing a more *optimal* PMS:

41. **Y**: *[Having completed her first valid PMS]* Now, I want to simplify it (Figure 10a).

[Teacher Y restructured the codes by nesting the loop; see Figure 10b.*]*

42. **R**: What is the function of the last two blocks [move 100 steps, turn left 72 degrees]?

43. **Y**: It is used to draw the pentagon. I try to throw it out.

[Teacher Y restructured the codes by removing the blocks of [move 100 steps, turn left 72 degrees] and the outermost loop (Figure 10c).*]*

From the process described above, we infer that Teacher Y *restructured* her codes twice. She first merged the two loops that were executed independently (one for drawing a pentagon and one for drawing squares and triangles), thus forming a nested loop structure (Figure 10b). At this point, the researcher noted that Teacher Y had placed the loop of drawing squares and triangles to be executed in front of the pentagon. When the researcher asked what the role of the last block was, Teacher Y noticed the recurrence of the two blocks [move 100 steps, turn left 72 degrees]. Upon this realisation that the preceding loops had sufficed to draw the pentagon, she reorganised the codes shown in Figure 10b by removing the outermost loop used to draw the pentagon and obtained a more concise PMS (Figure 10c). When elaborating on her code restructuring process, Teacher Y realised that the PMS could be further optimised:

44. **R**: I saw that you just thought of removing the outer pentagon. What do you think about this?

45. **Y**: [...] Whenever it draws a square, it will draw one more side. Moving 100 steps actually implies drawing a

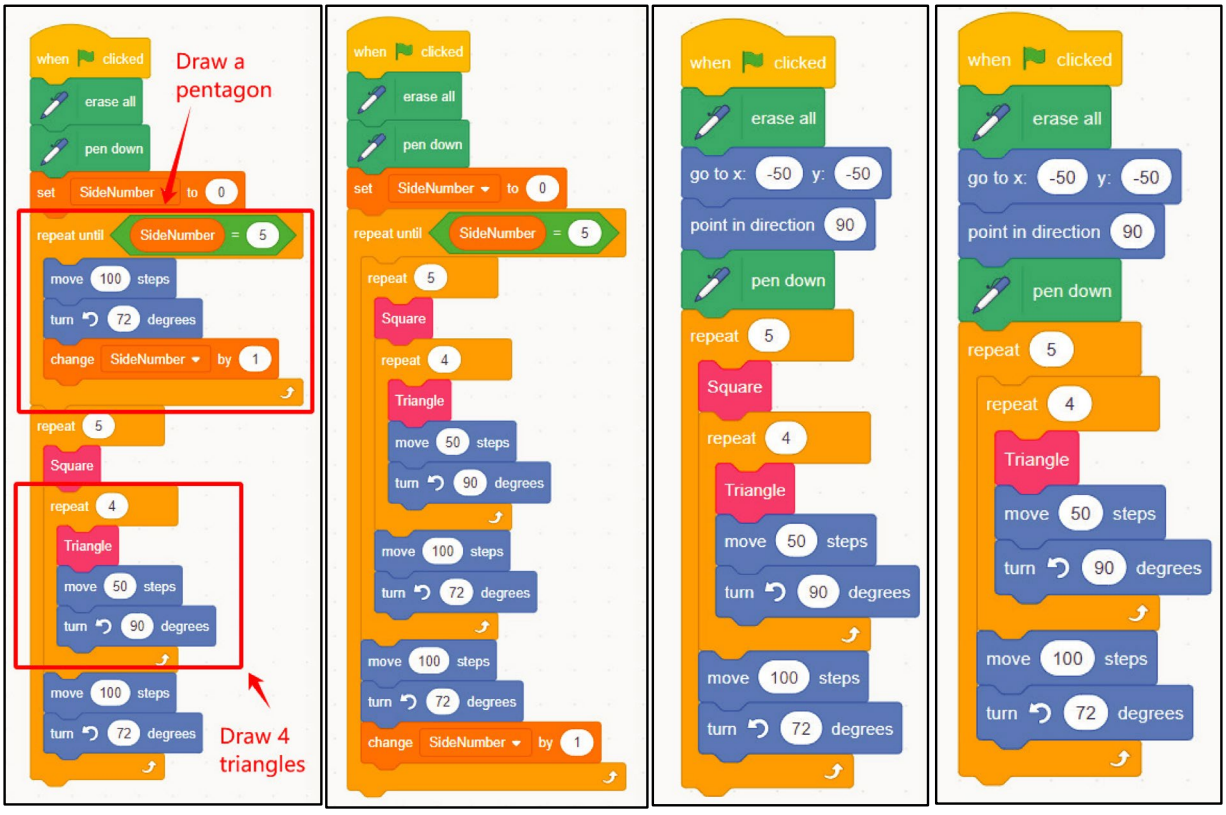


FIGURE 10 | (a) Teacher Y's first valid PMS; (b) Teacher Y restructured the PMS by nesting one loop into another; (c) Teacher Y restructured the codes by removing some blocks and the outermost loop; (d) Teacher Y's optimal PMS.

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

pentagon. If so, drawing a triangle seems to be repeated here... It seems like I don't need to use this to define a square (subroutine). It seems like I only need to define the triangle.

[Teacher Y restructured the code by removing the square subroutine (Figure 10d)*.]*

46. **R**: Can you explain what's going on now?
47. **Y**: It first draws a triangle from the position of (−50, −50). Then (the Pen) returns to the starting point, walks forward 50 steps, and turns left 90°. Turn the Pen 90° left and draw another triangle. Here, because I asked it to repeat the process four times, it drew a square. The loop embedded inside is used to draw a square. After drawing the square, we require that it return to the starting point, walk forward 100 steps, turn 72° to the left, and then repeat the triangle inside to draw a square. With this, it has to execute it five times, so the following is five (squares and triangles), and (the target figure) is drawn.
48. **R**: Do you remember your initial solution?
49. **Y**: Yes, I started by drawing a pentagon. I nested them layer by layer... so there would be repetition. I generated a new idea about the squares and triangles later and found that they would repeat together, so I simplified the latter ...

Teacher Y's explanation [Turn 45] implied that she was aware that drawing five squares would also consequently form a larger pentagon. Explaining this critical point to the researcher made it possible for her to associate an analogous process of drawing four triangles to form a larger square. As such, she suggested that the PMS shown in Figure 10c would not need to call on the square subroutine [Turn 45]. We argue that Teacher Y's optimisation of the original *suboptimal PMS* did not take place immediately but was constructed progressively after activating her CT and MT. In other words, MT underpinned her understanding of the polygon drawing process involved in the codes [Turn 47], and CT ensured her recognition of the nested process (i.e., she merged several parallel loops into a nested loop structure and removed the subroutine 'square'). Thus, the *restructuring* process provided opportunities for Teacher Y to gain a deeper understanding of the problem by orchestrating CT and MT.

5 | Discussion and Implications

Previous research has emphasised the necessity of examining CT as a dynamic process rather than isolating it as a specific concept or component (Kopcha and Ocak 2023). In alignment with this perspective, the current study underscores the importance of focusing on the developmental processes of CT and MT. Through an exploration of how problem solvers developed a PMS when they engaged in programming-based mathematical problem-solving, four pathways comprising seven distinct subsituations were identified. These findings are summarised in Figure 11. In the following, we further elaborate on the results, together with a discussion on the implications for theory and practice in mathematics education.

The first pathway begins with personal MT, in which problem solvers translate their mathematical knowledge into PMS within a programming environment. This translation process can manifest in various ways, and our study offers empirical evidence of the translation from personal MT to an invalid PMS. In our case, Teacher Z experienced difficulties with translating his personal MT into a valid PMS due to the inconsistencies between the languages of mathematics and programming (Cui and Ng 2021; Ye, Ng, and Cui 2023). Ye, Ng, and Cui (2023) emphasised the importance of developing translation flexibility in programming-based mathematics learning. Based on the current findings, we echo this and call on educators to identify

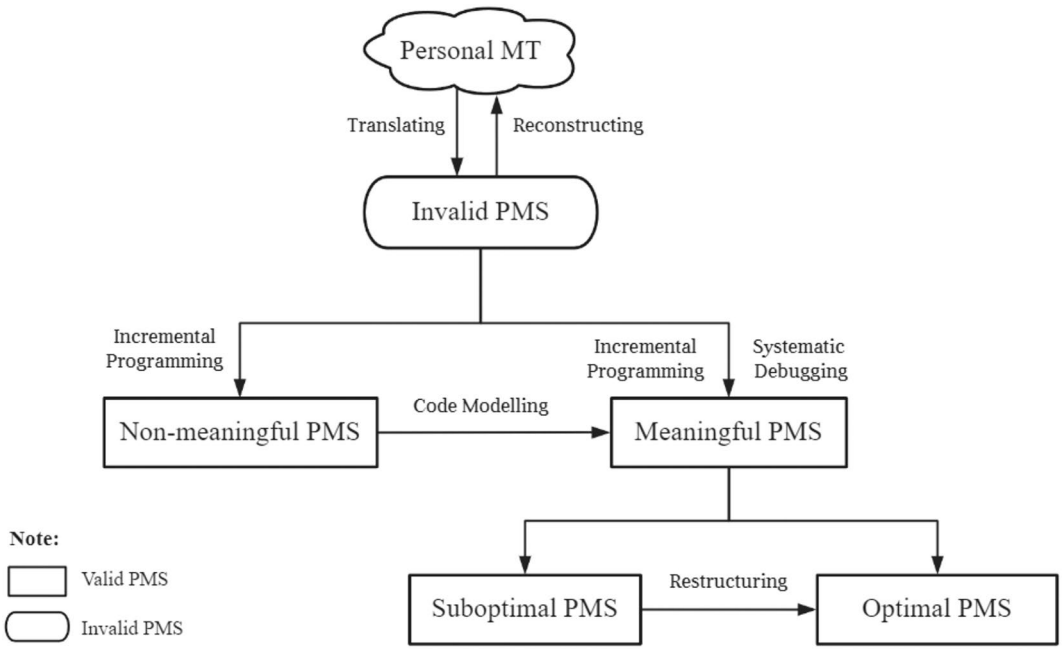


FIGURE 11 | Framework of pathways to develop programming-based mathematical solutions.

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

effective ways to support learners in addressing inconsistencies in the two learning environments. Dealing with these potential inconsistencies aligns with Wing's (2006) proposal that everyone should possess CT in the 21st century, which is the 'the mental activity in formulating a problem to admit a computational solution' (Wing 2017, 8). Meanwhile, we argue that these inconsistencies are not necessarily constraints or limitations but rather can be learning opportunities. For example, we found that problem-solvers are capable of reconstructing their MT to improve an invalid PMS.

The second pathway involves the transition from an invalid PMS to a valid PMS. Problem solvers often employ incremental programming or systematic debugging to change an invalid PMS into a valid one. Notably, incremental programming may yield meaningful PMS or non-meaningful PMS, which underscores the importance of attending to the extent to which problem-solvers make sense of their PMS beyond obtaining certain programming outcomes (Grover et al. 2017). Systematic debugging, in contrast, typically results in a meaningful PMS. This may be attributed to the problem solver's clarity in identifying the factors causing the invalidity of a PMS during the debugging process. Moreover, they engage in reflective practices by aligning their debugging methods with mathematical considerations. Consequently, problem solvers exhibit a clear understanding of the process involved in debugging the PMS. As described in the first pathway, despite the routine practices of testing and debugging in programming, there are instances in which problem solvers opt to terminate the debugging of invalid PMS and revisit their existing mathematical understanding. Thus, the different approaches to handling invalid PMS, whether terminating or persisting with systematic debugging, lead to distinct problem-solving trajectories. In this regard, one of the important questions is how problem solvers decide which way of handling should be adopted when faced with an invalid PMS? Specifically, what criteria guide their decisions to terminate debugging and revisit MT? When do problem solvers persist with systematic debugging instead of giving up in the face of challenges? These questions warrant further in-depth exploration in future research to gain insights into teaching practices.

The third pivotal pathway refers to the transition from a non-meaningful PMS to a meaningful PMS. A prevalent learning practice in transforming a non-meaningful to a meaningful PMS is engaging in code modelling to comprehend the elements of the PMS. Additionally, in Teacher Y's case, we observed the potential impact of self-narrating or articulating one's thoughts to others in this transformation process. Part of Teacher Y's sense-making of PMS was through slowly explaining her confusion about the valid PMS to the researcher, decomposing the code elements and describing the respective process and effects. This aligns, to some extent, with Sfard's notion of thinking as communicating (Sfard 2008), shedding light on the affordances of discourse in problem solving within programming environments.

Regarding the fourth pathway, the shift from a suboptimal to an optimal PMS is achieved by restructuring the codes, which can be understood from different theoretical perspectives. On one hand, from a flexibility perspective (Newton et al. 2020; Star and Seifert 2006; Ye, Ng, and Cui 2023), an optimal PMS reflects the problem solver's adeptness in flexibly transitioning between different PMS for the same problem when addressing programming-based mathematical problems. This involves their understanding of the problem context, flexibility in choosing and judging solution strategies and effective orchestration of CT and MT (see the notion of integrated flexibility in Ye, Ng, and Cui 2023). On the other hand, from the perspective of variation theory (Runesson 2005), 'a necessary condition for finding and experiencing a certain aspect of an object of learning is that the learner has the opportunity to encounter differences in the relevant dimension of variation' (Marton 2014, 128). Thus, we suggest that code restructuring may provide an opportunity for problem solvers to gain a deeper understanding of the knowledge, skills and thinking involved in addressing a problem by comparing the distinctions between different PMS. Taken together, we do not necessitate that problem solvers pursue the utmost simplicity or optimisation in the PMS. Instead, we suggest that the optimisation level of PMS is not the end of learning but rather a means for problem solvers to develop their CT and MT. In scenarios in which multiple PMS exist, problem solvers should be encouraged to activate and orchestrate their CT and MT to reflect on the existing PMS. The reconstruction of a suboptimal PMS into an optimal PMS has the potential to deepen their understanding of the problem and its solutions and advance their co-development of CT and MT.

Finally, we emphasise the investigation of the problem solver's learning process through the lens of solution development pathways as the theoretical contribution of this study. In comparison to other theoretical frameworks that focus on problem solvers' practices (e.g., Kaufmann and Stenseth 2021; Lee et al. 2011), the framework proposed in this study, grounded in the solution development stage, offers researchers and educators a structured approach to notice, discern and interpret students' learning and be responsive to their thinking in programming-based mathematics classroom (Steffe and Thompson 2000). From a practical perspective, these solution pathways not only reveal the diversity of learning processes in programming-based mathematical problem-solving but also serve as a valuable reference for instructional design. By leveraging these pathways, educators can design varied and scaffolded tasks to support students in achieving a comprehensive problem-solving process. Furthermore, the findings of this study can inform teacher professional development initiatives, addressing current significant gaps in existing professional development literature regarding responsive teaching in the context of programming-based mathematical problem-solving (Kyriakides et al. 2024; Shure et al. 2025; Zbiek et al. 2024).

6 | Limitations, Future Research and Conclusion

The findings of this study contribute to a deeper understanding of problem solvers' thinking in programming-based mathematical problem-solving and offer implications for theory and practice in K-12 programming-rich mathematics education. While situated within the K-12 mathematics education context, we acknowledge that this study is limited to a specific programming tool, a small number of cases and the specific knowledge background of the participants, which may limit the pathways and practices involved in the transition processes to a certain

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

number of situations. Various practices and strategies may exist for transforming to another state under the same solution state, and the pathways may not occur in a linear manner but rather a cyclical one. We thus call for future studies to continue investigating the diverse ways in which learners engage with programming-based mathematical problem solving and the challenges they encounter when transitioning between solution states.

Nevertheless, this study enhances our understanding of problem solvers' learning processes in solving programming-based mathematical problems. We identified different solution pathways and found that the transition between different solution stages provides opportunities for the development of CT and MT. It is imperative to underscore the significance of integrating CT and MT in programming-based mathematical problem solving, as this not only serves to generate new mathematical knowledge but also emphasises the necessity of constructing mathematics from a CT perspective. As an exploratory study, our findings provide a foundational framework for understanding solution pathways in programming-based mathematical problem-solving. However, further research with larger and more diverse samples is needed to strengthen and expand this framework. Therefore, future research is encouraged to explore the problem-solving process from diverse perspectives to develop a full picture of programming-based mathematical problem-solving.

Author Contributions

Huiyan Ye: conceptualization, investigation, writing – original draft, writing – review and editing, methodology, formal analysis, data curation, project administration, validation. **Biyao Liang:** validation, formal analysis, writing – review and editing, conceptualization, methodology. **Oi-Lam Ng:** conceptualization, methodology, validation, writing – review and editing, formal analysis, supervision, funding acquisition.

Acknowledgements

We would like to thank all the participants who took part in this study.

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

Aho, A. V. 2012. "Computation and Computational Thinking." *Computer Journal* 55, no. 7: 832–835. https://doi.org/10.1093/comjnl/bxs074.

Aminah, N., Y. L. Sukestiyarno, W. Wardono, and A. N. Cahyono. 2022. "Computational Thinking Process of Prospective Mathematics Teacher in Solving Diophantine Linear Equation Problems." *European Journal of Educational Research* 11, no. 3: 1495–1507.

Bhattacherjee, A. 2012. "Social Science Research: Principles, Methods, and Practices." USF Tampa Bay Open Access Textbooks Collection. Book 3.

Brennan, K., and M. Resnick. 2012. "New Frameworks for Studying and Assessing the Development of Computational Thinking." In *Proceedings of the 2012 Annual Meeting of the American Educational Research Association*, vol. 1, 25. American Educational Research Association. http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf.

Burton, L. 1984. "Mathematical Thinking: The Struggle for Meaning." *Journal for Research in Mathematics Education* 15, no. 1: 35–49. https://doi.org/10.2307/748986.

Cui, Z., and O. Ng. 2021. "The Interplay Between Mathematical and Computational Thinking in Primary School Students' Mathematical Problem-Solving Within a Programming Environment." *Journal of Educational Computing Research* 59, no. 5: 988–1012. https://doi.org/10.1177/0735633120979930.

Dickes, A. C., A. V. Farris, and P. Sengupta. 2020. "Sociomathematical Norms for Integrating Coding and Modeling With Elementary Science: A Dialogical Approach." *Journal of Science Education and Technology* 29, no. 1: 35–52. https://doi.org/10.1007/s10956-019-09795-7.

Grover, S., S. Basu, M. Bienkowski, M. Eagle, N. Diana, and J. Stamper. 2017. "A Framework for Using Hypothesis-Driven Approaches to Support Data-Driven Learning Analytics in Measuring Computational Thinking in Block-Based Programming Environments." *ACM Transactions on Computing Education* 17, no. 3: 1–25. https://doi.org/10.1145/3105910.

Grover, S., and R. Pea. 2013. "Computational Thinking in K-12: A Review of the State of the Field." *Educational Researcher* 42, no. 1: 38–43. https://doi.org/10.3102/0013189X12463051.

Hu, A., N. J. Hatfield, and M. D. Beckman. 2025. "Exploring Individuals' Computational Thinking With Data." *ZDM Mathematics Education* 57: 119–135. https://doi.org/10.1007/s11858-025-01669-0.

Hu, L., and H. Wang. 2024. "Unplugged Activities in the Elementary School Mathematics Classroom: The Effects on Students' Computational Thinking and Mathematical Creativity." *Thinking Skills and Creativity* 54: 101653. https://doi.org/10.1016/j.tsc.2024.101653.

Jonassen, D. H. 1997. "Instructional Design Models for Well-Structured and Ill-Structured Problem-Solving Learning Outcomes." *Educational Technology Research and Development* 45, no. 1: 65–94. https://doi.org/10.1007/BF02299613.

Kalelioglu, F., Y. Gulbahar, and V. Kukul. 2016. "A Framework for Computational Thinking Based on a Systematic Research Review." *Baltic Journal of Modern Computing* 4, no. 3: 583–596.

Kaufmann, O. T., and B. Stenseth. 2021. "Programming in Mathematics Education." *International Journal of Mathematical Education in Science and Technology* 52, no. 7: 1029–1048. https://doi.org/10.1080/0020739X.2020.1736349.

Kopcha, T. J., and C. Ocak. 2023. "Children's Computational Thinking as the Development of a Possibility Space." *Computers and Education Open* 5: 100156. https://doi.org/10.1016/j.caeo.2023.100156.

Kyriakides, L., C. Y. Charalambous, and P. Antoniou. 2024. "Integrating Subject-Generic and Subject-Specific Teaching Frameworks: Searching for Stages of Teaching in Mathematics." *ZDM Mathematics Education* 56, no. 5: 801–814. https://doi.org/10.1007/s11858-024-01591-x.

Lee, I., F. Martin, J. Denner, et al. 2011. "Computational Thinking for Youth in Practice." *ACM Inroads* 2, no. 1: 32–37.

Mannila, L., V. Dagiene, B. Demo, et al. 2014. "Computational Thinking in K-9 Education." In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference*, 1–29. ACM. https://doi.org/10.1145/2713609.2713610.

Marton, F. 2014. *Necessary Conditions of Learning*. Routledge.

Mason, J., L. Burton, and K. Stacey. 1985. *Thinking Mathematically*. Addison-Wesley.

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Mateus, A., H. Oliveira, and J. Piedade. 2024. "A Task for Integrating Computational Thinking in the Learning of Affine Function: An Exploratory Study With 8th Grade Students." *Quadrante* 33, no. 2: 193–222.

Miller, J. 2019. "STEM Education in the Primary Years to Support Mathematical Thinking: Using Coding to Identify Mathematical Structures and Patterns." *ZDM Mathematics Education* 51, no. 6: 915–927. https://doi.org/10.1007/s11858-019-01096-y.

Newton, K. J., K. Lange, and J. L. Booth. 2020. "Mathematical Flexibility: Aspects of a Continuum and the Role of Prior Knowledge." *Journal of Experimental Education* 88, no. 4: 503–515. https://doi.org/10.1080/00220973.2019.1586629.

Ng, O., A. Leung, and H. Ye. 2023. "Exploring Computational Thinking as a Boundary Object Between Mathematics and Computer Programming for STEM Teaching and Learning." *ZDM Mathematics Education* 55, no. 7: 1315–1329. https://doi.org/10.1007/s11858-023-01509-z.

Papert, S. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books Inc.

Pei, C., D. Weintrop, and U. Wilensky. 2018. "Cultivating Computational Thinking Practices and Mathematical Habits of Mind in Lattice Land." *Mathematical Thinking and Learning* 20, no. 1: 75–89. https://doi.org/10.1080/10986065.2018.1403543.

Polya, G. 1971. *How to Solve It: A New Aspect of Mathematical Method*. 2nd ed. Princeton University Press.

Repenning, A., and A. Basawapatna. 2021. "Smacking Screws With Hammers: Experiencing Affordances of Block-Based Programming Through the Hourglass Challenge." In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 267–273. ACM.

Rich, K. M., C. Strickland, T. A. Binkowski, and D. Franklin. 2019. "A K-8 Debugging Learning Trajectory Derived From Research Literature." In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 745–751. ACM.

Rodríguez-Martínez, J. A., J. A. González-Calero, and J. M. Sáez-López. 2020. "Computational Thinking and Mathematics Using Scratch: An Experiment With Sixth-Grade Students." *Interactive Learning Environments* 28, no. 3: 316–327. https://doi.org/10.1080/10494820.2019.1612448.

Runesson, U. 2005. "Beyond Discourse and Interaction. Variation: A Critical Aspect for Teaching and Learning Mathematics." *Cambridge Journal of Education* 35, no. 1: 69–87. https://doi.org/10.1080/0305764042000332506.

Selby, C. C., and J. Woollard. 2013. "Computational Thinking: The Developing Definition."

Sfard, A. 2008. *Thinking as Communicating: Human Development, the Growth of Discourses, and Mathematizing*. 1st ed. Cambridge University Press. https://doi.org/10.1017/CBO9780511499944.

Shure, V., M. Lehmann, M. Friesen, B. Roesken-Winter, and S. Prediger. 2025. "Professional Development Research on Promoting Productive Practices for Teachers' Handling of Typical Tasks of Mathematics Teaching: A Systematic Review." *ZDM Mathematics Education* 57: 795–813. https://doi.org/10.1007/s11858-025-01659-2.

Sinclair, N., and M. Patterson. 2018. "The Dynamic Geometrisation of Computer Programming." *Mathematical Thinking and Learning* 20, no. 1: 54–74. https://doi.org/10.1080/10986065.2018.1403541.

Stacey, K. 2006. "What Is Mathematical Thinking and Why Is It Important." In *Progress Report of the APEC Project: Collaborative Studies on Innovations for Teaching and Learning Mathematics in Different Cultures (II)—Lesson Study Focusing on Mathematical Thinking*, 39–48. ACM.

Star, J. R., and C. Seifert. 2006. "The Development of Flexibility in Equation Solving." *Contemporary Educational Psychology* 31, no. 3: 280–300.

Steffe, L. P., and P. W. Thompson. 2000. "Interaction or Intersubjectivity? A Reply to Lerman." *Journal for Research in Mathematics Education* 31, no. 2: 191–209.

Strauss, A., and J. Corbin. 1990. *Basics of Qualitative Research*. Sage Publications.

Tang, X., Y. Yin, Q. Lin, R. Hadad, and X. Zhai. 2020. "Assessing Computational Thinking: A Systematic Review of Empirical Studies." *Computers & Education* 148: 103798.

Tikva, C., and E. Tambouris. 2021. "Mapping Computational Thinking Through Programming in K-12 Education: A Conceptual Model Based on a Systematic Literature Review." *Computers & Education* 162: 104083. https://doi.org/10.1016/j.compedu.2020.104083.

Weintrop, D., E. Beheshti, M. Horn, et al. 2016. "Defining Computational Thinking for Mathematics and Science Classrooms." *Journal of Science Education and Technology* 25, no. 1: 127–147. https://doi.org/10.1007/s10956-015-9581-5.

Wiesner, C. 2022. "Doing Qualitative and Interpretative Research: Reflecting Principles and Principled Challenges." *Political Research Exchange* 4, no. 1: 2127372.

Wing, J. 2006. "Computational Thinking." *Communications of the ACM* 49, no. 3: 33–35. https://doi.org/10.1145/1118178.1118215.

Wing, J. 2011. "Research Notebook: Computational Thinking—What and Why." *Link Magazine* 6: 20–23.

Wing, J. 2017. "Computational Thinking's Influence on Research and Education for All." *Italian Journal of Educational Technology* 25, no. 2: 7–14.

Ye, H., B. Liang, O. Ng, and C. S. Chai. 2023. "Integration of Computational Thinking in K-12 Mathematics Education: A Systematic Review on CT-Based Mathematics Instruction and Student Learning." *International Journal of STEM Education* 10, no. 1: 3. https://doi.org/10.1186/s40594-023-00396-w.

Ye, H., O. Ng, and Z. Cui. 2023. "Conceptualizing Flexibility in Programming-Based Mathematical Problem-Solving." *Journal of Educational Computing Research* 62, no. 2: 374–399. https://doi.org/10.1177/07356331231209773.

Yeung, W., O. Ng, and Y. Zhang. 2024. "Young Children's Embodied Computational Thinking Developed With Touchscreen Mathematics Applications." *Quadrante* 33, no. 2: 11–35.

Zbiek, R. M., S. A. Peters, B. Galluzzo, and S. J. White. 2024. "Secondary Mathematics Teachers Learning to Do and Teach Mathematical Modeling: A Trajectory." *Journal of Mathematics Teacher Education* 27, no. 1: 55–83. https://doi.org/10.1007/s10857-022-09550-7.

Zhang, L., and J. Nouri. 2019. "A Systematic Review of Learning Computational Thinking Through Scratch in K-9." *Computers & Education* 141: 103607.

13652729, 2025, 5, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.70102, Wiley Online Library on [17/11/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License