FLOWGORITHM: A possible passage from algorithmic reasoning to creatively founded mathematical reasoning

Weng Kin Ho[*†] wengkin.ho@nie.edu.sg
Chee Kit Looi[*] cheekit.looi@nie.edu.sg
Wendy Huang[*] wendy.huang@nie.edu.sg
Peter Seow[*] peter.seow@nie.edu.sg
Shiau Wei Chan[*] shiauwei5634@gmail.com
Longkai Wu[*] longkai.wu@nie.edu.sg
National Institute of Education
Nanyang Technological University
Singapore

September 12, 2021

Abstract

This paper argues that algorithm design in the sense of computational thinking (CT) does not involve only routinized procedural applications void of deep conceptual understanding of mathematics. By using the programming language FLOWGROITHM*, we demonstrate how classroom tasks centred around algorithm design may be used to activate creatively founded mathematical reasoning (CMR) in mathematics students.*

1 Introduction

1.1 Mathematical competencies and abilities

The mathematics teacher's primary role is to help students develop their *mathematics competencies*, i.e., abilities to "understand, judge, do, and use mathematics in a variety of intra- and extra-mathematical contexts and situations in which mathematics plays or could play a role" [12, pp. 6-7]. In particular, [10] expanded mathematics competencies into six *abilities*: (1) problem solving ability, (2) reasoning ability, (3) applying procedures ability, (4) representation ability, (5) connection ability and (6) communication ability.

[*]This work is supported by funding OER 10/18 LCK for the project "How to bring Computational Thinking (CT) into Mathematics classrooms: Designing for disciplinary-specific CT".

[†]Corresponding author

Worldwide, classroom tasks are thus designed to develop these six abilities. This paper proposes the use of algorithm as nexus connecting the abilities of problem-solving (how to solve tasks without knowing a solution method in advance), reasoning (the ability to justify choices and conclusions) and applying procedures (the ability to execute known procedures that are learnt by heart).

Applying procedures is familiar to mathematics teachers. In mathematics and mathematics education, such procedures are called *algorithms*. An *algorithm* is a set of rules to be followed in calculations or other problem-solving operations, be it by a machine or a human being. In mathematics education, this definition is broadened below:

Definition 1 (Algorithm) *An algorithm is a set of finite sequences of executable instructions enabling one to solve a given set of tasks ([3]).*

Common algorithms encountered by Singapore secondary school students (aged from 13 to 16 [1]) in the Singapore Mathematics syllabuses ([11]) include, but are not limited to, multiplication and long division (involving integers and/or polynomials), prime factorisation, obtaining highest common factors and least common multiples of natural numbers, quadratic formula, mensuration formulae, Pythagoras Theorem, plotting graphs, using formulae in coordinate geometry (e.g., gradient of line segment, length of a line segment, area of a triangle the vertices of which have given coordinates), partial fractions decomposition, finding derivatives and antiderivatives of special functions, etc. In short, the ability of applying algorithms in mathematics is tied on to routinized procedures. Imagine a continuum where a mathematical task is placed based on the availability of known mathematical procedures to the student engaged in that task. Then a task which merely requires the student to invoke a known algorithm (i.e., to mimic the teacher's demonstration of the same algorithm) will be placed on one end of this spectrum.

Thus, problem-solving tasks are placed on the other end of this spectrum. By a *problem*, we mean a task in which a student who is engaged in solving it does not have a readily available method or procedure to solve it. According to [8, p. 50], successful problem solving involves coordinating previous experiences, knowledge, familiar representations and patterns of inference, and intuition in an effort to generate new representations and related patterns of inference that resolve some tension or ambiguity (i.e., lack of meaningful representations and supporting inferential moves) that prompted the original problem-solving activity. A problem solver inevitably engages in the Pólyan cycle of understanding the problem (UP), devising a plan (DP), carrying out the plan (CP) and checking and/or extending the solution (CE) ([13]).

1.2 Concerns about extensive use of algorithm

Several studies ([2, 4]) have revealed that the majority of the class-time had been spent on learning and rehearsing algorithms which, when mastered, are intended to ensure students' efficiency and accuracy in solving mathematics problems and tasks that they tackle at some later stage. Algorithms form an essential body of knowledge for mathematics students because they are meant to yield quick and reliable answers to those tasks that can be solved by following a 'fixed' solution path. Because situations where a certain algorithm can be employed are completely deterministic and predictable, tasks that involve merely repeated usage of the

[1] Typically, Secondary One students are aged 13, Two 14, Three 15 and so on.

algorithm then provide students plenty opportunity to practise. In themselves, there is nothing innately wrong about tasks that involve students' practice on applying algorithms but it is the extensive use of such tasks, that some studies, suggest may be counterproductive [4]. The danger here occurs when a classroom situation is typified by (a) a teacher providing students a set of mathematical tasks that have a common solution template, and (b) students repeatedly applying the same algorithm in solving this set of tasks. Then the students would have no opportunity to reflect on their use of the algorithm. It is this unreflective use of the algorithm which detaches the meaning of the algorithm from its application ([3]).

In summary, if there is an unchecked and extensive use of algorithmic learning that does not invoke the learner's reflection about the meaning of the algorithm then there is concern whether or not the learner's mathematical reasoning abilities will hampered.

1.3 Proposed approach

The problem lies not with algorithmic learning but instead the way classroom tasks are designed (and implemented). In this paper, we suggest various ways whereby algorithms can in fact be exploited to create opportunities for deep conceptual learning of mathematics and active engagement in problem solving. This approach gives a counter-argument against the usual acceptance that algorithmic learning deprives student of creatively founded mathematical reasoning.

2 Algorithm design in Computational Thinking

2.1 Algorithmic and creatively mathematically founded reasonings

Two types of reasoning are specifically mentioned in [10, 9]: *algorithmic reasoning* (AR) and *creative mathematically founded reasoning* (CMR). The notion of AR, first coined by [10], refers to the state in which a problem solver employs repetitive numerical task-solving method – such a method must make use of an algorithm that can be employed to solve the problem (and one which is provided together with the task). As for CMR, [9] defines it to embody all of the following attributes: (a) *creativity* – a reasoning sequence new to the reasoner is created, or a forgotten one is re-created; (b) *plausibility* – arguments supporting both the strategy choice and implementation must be there to provide plausible or correct reasons to reach the conclusion(s); and (c) *anchoring* – placing arguments at the intrinsic mathematical properties of the components that are involved in the reasoning required to solve the problem.

Regarding AR and CMR, two significant findings reported in a recent work ([7]) by B. Jonsson and his team motivate our current approach: (1) AR leads to better students' performance during practice sessions when compared to CMR, as a function of the algorithmic support that was provided in the task (i.e., solution method and/or template is available to the students when given the task). (2) CMR outperforms AR whenever a task involves (i) conceptual understanding, (ii) memory retrieval and/or (iii) (re)construction of solution methods.

From the above finding (1), we gather that AR leads to positive learning outcomes during the initial stage of student's familiarization with key definitions, methods, calculations, algorithms and solution templates through practice. However, finding (2) informs us that the learner must move on from this initial stage of drill-and-practice to engage with (cognitively) more

demanding tasks that require genuine conceptual understanding, retrieval of useful information and/or (re)construction of solution methods.

2.2 Algorithm design

In an earlier work [6] presented at the 24th ACTM, the authors gave domain-specific interpretation for the four components of Computational Thinking (CT), namely, Decomposition, Pattern Recognition, Abstraction and Algorithm Design. Additionally, they argued for the use of task design principles grounded in Computational Thinking (CT) to construct meaningful tasks which exploit CT in mathematics learning.

The specific component of Computational Thinking we are focusing in this paper is *Algorithm Design*, which according to that same paper,

> "involves the planning and development of a set of precise and step-by-step instructions for solving the problem."

Such a finite sequence of instructions is termed as a *program* which can either be carried out by a computer or a human being in an 'insightless' manner. Notice here that running the algorithm or program requires no insight and hence unlikely to result in meaningful learning. Indeed, a proficient use of algorithms can at best reduce both the cognitive demands of complicated calculations ([5]) and the cognitive load on the learner's working memory ([15]). But *planning and development* of such an algorithm requires not only insight but a lot of creativity and deep conceptual understanding of the mathematics that learners are engaging with.

Clearly, it is the way tasks are designed to train learners in planning and developing algorithms with the intention of activating students' reasoning that really matters. In fact this view is not new as already [9] reported that the reasoning that students activate as a result of active planning and developing (in relation to specific tasks) is one key variable in learning mathematics through task solving.

2.3 The role of algorithm design

This is where the element of Algorithm Design lend itself to transit the learner smoothly from the initial stage of applying the algorithm or solution template to the more cognitively demanding engagement with the higher-order thinking processes. To operationalize this transition, there are two distinctive task labels associated to the task we design around the element of Algorithm Design. (1) Implementing a procedure: Present the key algorithm/solution template in a *standard* format that can be communicated by the teacher, and can be understood by the student. (2) Problem solving: Construct (or co-construct) a novel algorithm which is used to solve a problem embedded in the mathematical task.

In order that a standard format be established to allow communication between the teacher and the student, we have chosen a free programming language called Flowgorithm. This software requires little overhead in programming knowledge. It allows its user to construct a program in the form of a flowchart and – the best part – to run the program.

Typically, in an 'implementing a procedure' task the teacher will display a flowchart program (that implements some mathematical procedures) for students to read. This is in line with the pedagogical approach called *PRIMM* ([14]), that is, (P) Predict: look at a short program and

try to guess what it will do; (R) Run: run the codes of the program, observe the outcome and check whether the prediction made earlier is correct; (I) Investigate: carry out investigations about the different part of the codes, e.g., tracing the program flow, making annotations, etc.; (M) Modify: changing the code to effect desired changes; (M) Make: make a brand new program by making use bits of the codes from the original program to solve a new problem. We shall explain the details of how the transition takes place from 'implementing a procedure' to 'problem solving' in the next section.

3 FLOWGORITHM as passage from AR to CMR

3.1 Flowchart and FLOWGORITHM

When a student first learns a text-based programming language, it is not uncommon that he or she be required to enter several lines of syntax (i.e., commands) before something simple can be achieved, e.g., to print "Hello, world!". One advantage of the using flowchart is to take away the cognitive load of familiarizing with the nuances of the text-based programming language, and let the learner focus on the visual flow of information, and thereby acquire a quick understanding of the underlying programming concepts. To print out "Hello, world!", the important point here is appreciate that "Hello, world" is a string (i.e., literally a string of characters that can be input through the keyboard) to be *output*. Figure 1 depicts the FLOWGORITHM program (on the left) to print in the chat bubble "Hello, world!" (on the right).

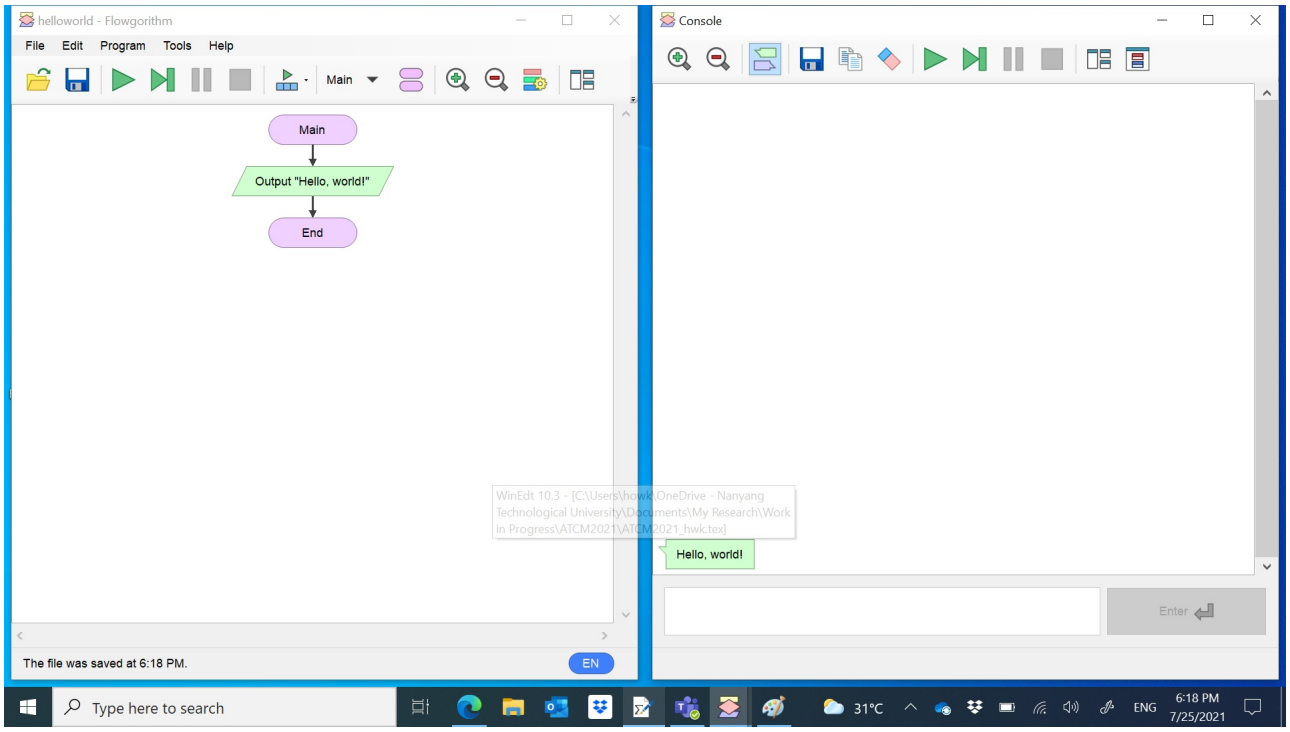

Figure 1: FLOWGORITHM program `helloworld.fprg`

On the official website `http://www.flowgorithm.org/download/index.htm`, one can find

the installation instructions for FLOWGORITHM, together with an introduction to how FLOWGORITHM programs are written.

3.2 Looking for the right topics

To illustrate how the aforementioned transit can be made from AR to CMR in an authentic classroom situation, we invoke a running example through this section. The first point to be made here is that it is important to identify the topics in which algorithms already reside.

The running example we have chosen sits in the topic of quadratic equations and expressions, which in the mathematics syllabus usually taught to Secondary Three (Express) students. We give some background of this topic below.

When students first encounter quadratic equations in Secondary Two, they will be taught how to recognize one, i.e., the students would check that a quadratic equation has only one variable, say x, and is of the form $ax^2 + bx + c = 0$, where a, b and c are constants and crucially $a \neq 0$. Usually these coefficients are given as integers, though not necessarily so.

Students are expected to be able to apply their prior knowledge in algebraic manipulation to reduce equations to a quadratic one, e.g., to be able to prove that the equation $\frac{10}{x} - \frac{10}{x+2} = \frac{1}{5}$ can be reduced to $x^2 + 2x - 200 = 0$.

At Secondary Two, students would be taught how to factorize a quadratic expression into its linear factors (over the polynomial ring $\mathbb{Z}[x]$) using trial-and-error. The method is known as the 'cross-method'. For instance, to factorize $2x^2 + x - 3$ into its linear factors, a student tries all possible integer factorization of the constant term "-3" (e.g., 3×-1 and -3×1). Figure 2 shows a typical working carried out by a student when employing the "cross-method". Notice that cross-multiplying x-terms with the factors yield $+3x$ and $-2x$ in the add-column on the right, where the sum of $+3x$ and $-2x$ is checked to be $+x$, as desired.

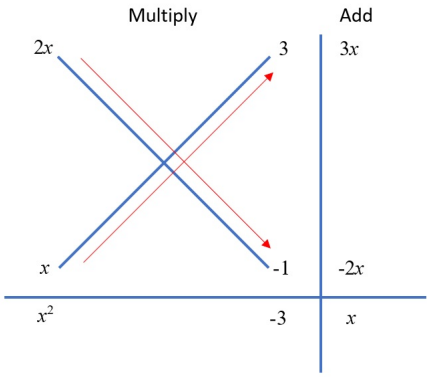

Figure 2: "Cross-method" for factorization

At Secondary Two level, students would realize that not all quadratic expressions can be factorized into linear factors with integer coefficients by solely applying the "cross-method". They are not expected to know the necessary and sufficient conditions for a quadratic equation with integer coefficients to be factorized into two linear factors with integer coefficients. At that level, the nature of roots, i.e., real or complex roots, is also not in the syllabus. According to the Secondary Mathematics Syllabus ([11]), students at Secondary Three are expected to

apply the quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1}$$

to obtain the roots of the quadratic equation $ax^2 + bx + c = 0$, assuming the real roots exist. Though the formula is available in the formula booklet, it is often advised that students memorized the formula and apply it at will. In a typical lesson based on the AR approach, the teacher will demonstrate the solution of a quadratic equation using a solution template typified in Figure 3.

In the equation $x^2 + 2x - 200 = 0$, the coefficients are $a = 1,\ b = 2,\ c = -200$.
Applying the quadratic formula,

$$\begin{aligned} x &= \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \\ &= \frac{-2 \pm \sqrt{2^2 - 4(1)(-200)}}{2(1)} \\ &= -11.06 \text{ and } 9.06, \end{aligned}$$

correcting the answers to two decimal places.

Figure 3: Solution template for using the quadratic formula

3.3 Starting with flowchart

The teacher can set a few AR-tasks for the student to drill-and-practice and familiarize with the above solution template. With the goal of transiting to the use of CMR approach in teaching and learning, we do not stop here. We suggest that the teacher introduces what an algorithm is in the sense of Definition 1. This can be done by inviting students to examine the structure of the solution template in Figure 3, using the following prompt:

Teacher: In the solution template that we have been using to solve those quadratic equations,

1. identify those key input data you must use to calculate the roots of the given equation $ax^2 + bx + c = 0$;
2. write down a step-by-step set of instructions using those data to output the required roots.

In the ensuing conversation between the teacher and the students, the discussion will likely culminate with the following points:

- Input(s): The key input data identified are the values of a, b and c – but not x.

- Process(es): Although the mathematical syntax employs a single variable x to denote the two possible roots, the roots are actually calculated using the input data and two separate formulae:
$$\frac{-b-\sqrt{b^2-4ac}}{2a} \text{ and } \frac{-b+\sqrt{b^2-4ac}}{2a}.$$
- Output(s): The first root can be stored as x_1, and the second as x_2.

The students' realization that there are in fact two outputs x_1 and x_2 deepens his or her understanding of the quadratic formula, where, in particular, the sign '$\pm$' really denotes two different operations (plus and minus) performed separately. Apart from this realization, one of the end-goals of the above task is to heighten the student's awareness to the 'Input-Process-Output' sequence in an algorithm, i.e., given some inputs (declared by the programmer, and later supplied by the user), some processes are then invoked to act on the supplied inputs, and these will finally yield some outputs. This observation then lends itself naturally into a flowchart – the paradigm of information-flow from the input source, through the processes (i.e., computations), to the output target.

At this point of the discussion, a flowchart that enacts the algorithm of calculating the two roots of a quadratic equation is displayed (see Figure 4).

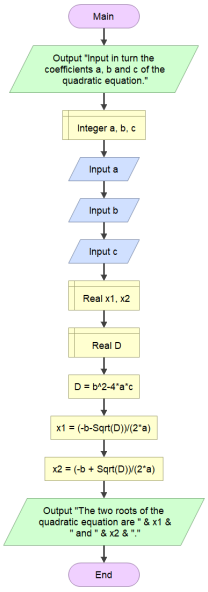

Figure 4: Flowchart for calculating the roots x_1 and x_2

The follow-up activity that makes use of the flowchart in Figure 4 would involve the students to predict what the flowchart does (given that they do not know some of the syntax in the flowchart) and communicate the intention of the key inputs, processes and outputs appearing in the flow of information from Main (input) to End (output). This activity is identified with the 'Predict' phase of the PRIMM framework.

The purpose of this segment is for the teacher to use the diagrammatic nature of the flowchart to explain what the flowchart does, and simultaneously introduce the fundamental computing concepts of variables as placeholders for storing values, e.g., `a` `b` and `c` have been set as variables taking on integer values, for example; and `x1` and `x2` are variables taking on

real number values. Additionally and concurrently, the programming concept of data and date types can be introduced to the students within the disciplinarity of mathematics.

3.4 Using FLOWGORITHM to build flowcharts

The next teaching move would be to show how FLOWGORITHM can be employed to build the flowchart shown in Figure 4. This demonstration will in addition introduce the following essential concepts: (1) String data type, e.g., the words appearing within the double inverted commas, and the concatenation operator "&". (2) Assign a value to a variable, e.g., defining `D`, `x1` and `x2` using formulae. (3) Re-use codes or variables in the program to avoid redundant repeated computations, e.g., calling up the variable `D` and using it later in `x1` and `x2`.

Each of these teaching points will involve different actions (e.g., Input, Declare, Assign, Output) which are distinguished by the different shapes. The directed arrows reinforce the concept of information-flow.

3.5 Running the flowchart in FLOWGORITHM

The role of the machine enters the scene when the teacher demonstrates that the constructed flowchart in Figure 4 is in fact *executable*, i.e., it can in fact be operated as a computing machine! Pressing the play button sets the program running, i.e., (i) the program interacts with the user by asking for inputs to `a`, `b` and `c`, and (ii) the program returns with the output embedded in the sentence "The roots of the quadratic equations are ... (first root `x1`) and ... (second root `x2`)". Figure 5 (the console on the right) shows an instance of running the flowchart in Figure 4 with the inputs $a = 1$, $b = 2$ and $c = -100$ as in the quadratic formula.

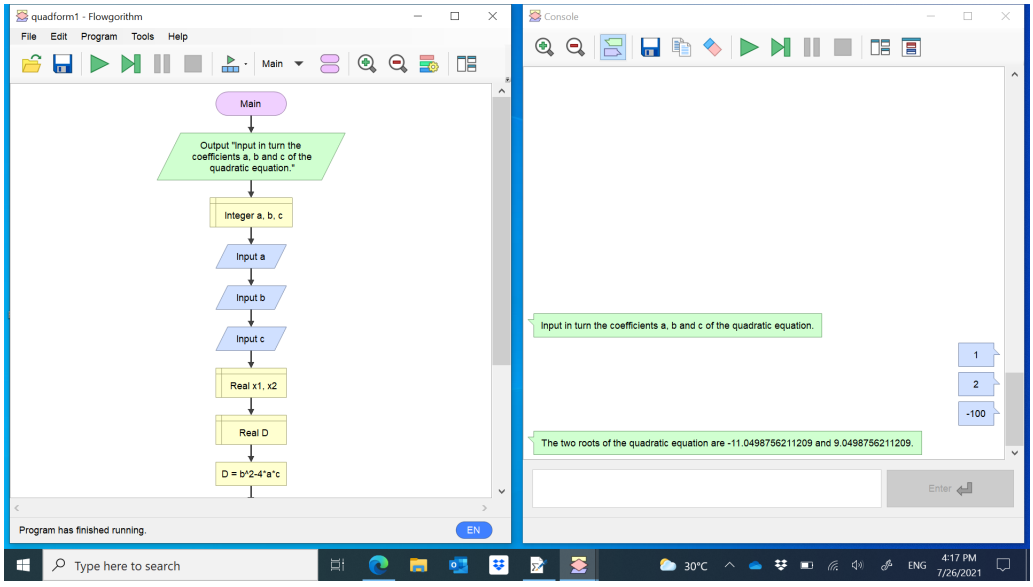

Figure 5: Running the flowgorithm program `quadform1.fprg`

The teacher may now invite the students to use their flowchart as a quadratic formula calculator to check that the answers obtained in the AR-tasks completed a priori are indeed correct. The end-goal of writing a FLOWGORITHM program in this lesson is to culminate with

a useful product, i.e., a calculator that can calculate the roots of a quadratic equation. Thus, this task is aligned with Papert's original conception of what Computational Thinking should achieve, i.e., the learner acquires knowledge by constructing a useful product (see [16, 6]).

We classify the above task as an "Implementing a procedure"-task because the main objective of this type of task is to get the students to appreciate that the solution template associated with solving a quadratic equation can in fact be implemented as an algorithm (in this case using FLOWGORITHM as the programming language). Additionally as the teacher walks the students through the flowchart (in Figure 4), the students are given another chance to internalize the quadratic formula on top of the drill-and-practice tasks completed prior to this flowchart activity. Also, a pleasant side-effect of this approach is that students acquire the necessary programming concepts through the use of FLOWGORITHM in the context of solving quadratic equations *without* being burdened by the syntax of a text-based programming language.

3.6 Moving on to problem solving

We recall from our earlier discussion in Section 3.2 that Secondary Three students "are not expected to know the necessary and sufficient conditions for a quadratic equation with integer coefficients to be factorized into two linear factors with integer coefficients." This now can be turned into an opportunity for the students to engage in authentic problem solving.

Moving forward, the teacher can craft the problem-solving task as follows:

> **Problem.** Given a quadratic equation $ax^2 + bx + c = 0$, with integer coefficients, i.e., a, b and c are all integers, devise a test involving a, b and c that would help us determine whether this equation can be solved by using the 'cross-method' (i.e., guess-and-check in the sense illustrated in Figure 2) involving integer coefficients for the linear factors.

Although the problem stated is not unfamiliar to students at this level, they do not have a ready solution to it. The teacher can guide the students through the flowchart and ask them to identify which part of the program is instrumental in calculating the roots of the equation. In other words, we are engaging the students in the 'Investigate' phase in the PRIMM framework. Suitable scaffolding can be designed to lead the students to examine the different sub-expressions or terms appearing in the quadratic formula.

The students can employ the FLOWGORITHM program (i.e., `quadform1.fprg`) shown in Figure 5 on several sets of integral values of a, b and c to explore the outcomes. A closer examination then reveals that the discriminant $D := \sqrt{b^2 - 4ac}$ must be a perfect square in order that $\sqrt{D} = \sqrt{b^2 - 4ac}$ is an integer so that the final expression $\frac{-b \pm \sqrt{D}}{2a}$ is rational.

For the 'Modify' or 'Make' phase, the teacher can continue to challenge the students to devise a FLOWGORITHM program to determine whether a given quadratic equation has rational roots.

4 Task design schema

In the preceding Section (i.e., 3.2–3.6), the sample tasks sequence is built around a selected mathematics topic, and algorithm design is embedded into the different tasks that help students transit from Algorithmic Reasoning (AR) to Creatively Mathematically Founded Reasoning

(CMR). We now abstract the details of this tasks sequence in the form of a task schema. Recall that schemas are categories of information stored in long-term memory ([1]). In other words, a schema contains groups of linked memories, concepts or words. The purpose of having a task design schema, in our case, is to create a cognitive shortcut for the task designer so that the specific task design we recommend here can be stored and retrieved from one's long-term memory much more quickly and efficiently. Because the classroom task is designed for teaching and learning of Mathematics via Computational Thinking, we employ the four task design principles proposed in [6]. For the reader's convenience, we briefly recall what these four design principles are, and then apply them as we describe the task schema proposed herein.

4.1 Four task design principles

The original intention of bringing in this set of task design principle is to address teacher's concern about instructional design centred around crafting Mathematics lesson that activate Computational Thinking. Because Computational Thinking comprises four key components, i.e., *decomposition*, *pattern recognition*, *abstraction* and *algorithm design*, the four design principles have bee targeted to address each of these components in the form of questions to which the task designer must answer. Below, we quote these questions verbatim from [6, pp. 5-6].

Complexity Principle. "Does the mathematical concept give rise to sufficiently complex problem?" The problem should involve the use of the identified concept, and be complex enough so that decomposition of this main problem into sub-problems is a needful step. If the problem or task is routine or too simple, e.g., there exists a ready-made solution or method, then decomposition is uncalled for.

Data Principle. "Can the mathematical concept occur in various forms so that it is possible to collect data for its occurrence?" The topic should involves observable and quantifiable data that can be collected, created, analyzed, and shared.

Mathematics Principle. "Can the problem associated to the mathematical concept be mathematized?" Mathematization is the formulation of the problem using mathematical terms. It turns a problem in real world context in an abstract and precise manner to a mathematics problem. We do not restrict mathematics to mean only numbers, algebra, geometry, and so on. Rather, mathematics can have a more inclusive meaning of encompassing abstract concepts and structures which are definable, representable, and can be reasoned about within some logical framework.

Computability Principle. "Does there exists an effectively calculable solution to the mathematized problem?" By 'effective calculable' we use it in the sense of Recursion (or Computability) Theory, that is, there exists a a computer program that can calculate a solution to the problem through a finite procedure via a physical agent (e.g., machine, human being).

It is however to be noted that there is *no* need for the above principles to be applied in any prescribed order.

4.2 Task schema proposed and validated

From the preceding section, we see that there are four key stages involved, and these are presented in the form of a schema shown in Figure 6.

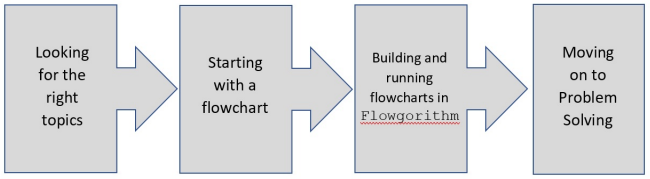

Figure 6: Task design schema for transiting from AR to CMR

Looking for the right topics. Since we are advocating the situating of algorithms as the connecting piece between AR and CMR, the task designer should naturally browse through the potential topic and look out for all the existing procedures, solution templates to standard questions, standard calculations involved, etc.

We now apply the *Computability Principle* at this juncture to check if those identified procedures or solution templates can be implemented as an algorithm. Even when these procedures are programmable, there is the question of whether the programming concepts or techniques are too difficult for the students at that level. In the present proposal, we recommend the use of FLOWGORITHM as a simple programming language with a low overhead of programming syntax. The question here is whether the algorithm when implemented in this language is simple enough for students of that level to understand.

Starting with a flowchart. At this stage, the student would be engaged in understanding what a flowchart involves, e.g., the key variables and their associated data types. In other words, the student becomes cognizant of the different components of the problem or algorithm at hand. This step of starting with a flowchart helps the student stay focused on the different inputs, the various decision-making and the calculations together with all the variables. As the student moves on from AR to CMR, he or she would be expected to be more independent in deciding which are the variables to be defined, and what data types they belong. All these are possible if the task designer factors in the process of Decomposition, i.e., breaking down the complex task into smaller sub-tasks which are more tractable. Thus, the *Complexity Principle* validates this part of the task schema in that this task engages the students in identifying the various parts of the flowchart – the students are moved towards appreciating this part-whole relationship between the individual parts of the algorithm (e.g., variables, sub-processes) and the whole algorithm itself. In addition, *Data Principle* can be applied here to ask for evidence of students' working out which input data are used and how these are transformed by the subsequent information-flow directed by the flowchart.

Building and running flowcharts in FLOWGORITHM. This is closely tied to the component of Algorithm Design in Computational Thinking, and we are thus invoking the *Computability Principle* again. At this stage, we check whether the task requires that the algorithm can be effectively implemented in the chosen language. In this part of the task schema, the students must

construct the given flowchart using FLOWGORITHM. Like any other programming language, students at this stage are bound to make mistakes and hence we expect them to be debugging their programs. While training them to pay attention to each of the constituent parts of the entire algorithm, we are indirectly helping the engaged students to internalize the algorithm. The authors are in the view that algorithmic learning implemented through this approach is far more meaningful than a mere memorization of formulae or routinized procedures.

Moving on to problem solving. Remembering that Computational Thinking is evidence-based because there are clear deliverables and observables tied to it – meaningful learning yields useful products. At the end of the AR-focused segment of the lesson, a useful product is obtained – the resulting FLOWGORITHM. Now the transit from AR to CMR crucially hinges on how the task requires the student to make use of the program that he or she has just constructed. In order that CMR be invoked, the activities at this stage must involve the learner to think in a deeper manner through the mathematical concepts involved. In our running example, the FLOWGORITHM program that implements the quadratic formula serves as an object of study in itself: which part of the formula determines the rationality of the roots of the equation given that all the coefficients are integers? The FLOWGORITHM program has the advantage that it can be run on input data so as to create visible patterns for students to engage in *P*attern Recognition and hence to formulate relevant conjectures towards solving the problem at hand. Hence we see that the task is centrally rooted in a mathematized problem – the *Mathematical Principle* thus validates this last but critical part of the task schema. Indeed without this part, the transition from AR to CMR can never take place. This is where the students' mathematical creativity emerges out of the situationally induced need to reasoning out for themselves what was going on in the heart of the algorithm.

5 Discussion

Having presented the task design schema that is targeted to transit the learner from AR to CMR, it is time to reflect on our proposed approach. Let us begin with some limitations.

5.1 Limitations

Flowcharts, even in the present form written in FLOWGORITHM, are hard to modify. Although the flow structure appears easy to understand, a working program is often completed *all at once* – one can hardly modularize by making the small parts work first. Of course, FLOWGORITHM allows one to call up functions (which can be coded separately from the main program) but it becomes intractable once the program involves too many function calls. Additionally, debugging can be challenging because one just cannot take apart the flowchart and reassemble the parts again. Our proposed approach relies heavily on the choice of the right topics. Even when there are fixed procedures, e.g., finding shortest distance of a point from a line requires one to identify the foot of perpendicular from the point to the line, sometimes there is such a great variety of situations (in which the same procedure is to be applied) that it is simply not possible to code the procedure as a computer program. This is especially the difficulty encountered when one is dealing with geometry problems.

5.2 Implications

Although the task design schema looks largely plausible to apply, it is far from being clear how the mathematics teacher can use it to generate specific lesson tasks on specifically selected topics. For example, can one incorporate this proposed approach systematically into a professional development program for mathematics teachers? At the moment of writing this paper, the first author is in the process of introducing the proposed approach of using algorithms to mathematics teachers at an in-service training program that is intended to equip them with basic programming skills. In this course, one special feature of FLOWGORITHM is frequently emphasized and exploited, i.e., all programs written in FLOWGORITHM can be exploited as programs written in a variety of different programming languages (e.g., C++, Fortran, Python, Javascript, etc.). From a practitioner's point of view, what one needs is a rich collection of exemplars, be they flowcharts of all the commonly encountered algorithms that are associated to implementing a procedure or those used to encourage problem solving or making conjectures. The first author, together with some beginning teachers, is building up a library of FLOWGORITHM (as well as others) programs for the purpose of teaching mathematics at the secondary level.

6 Conclusion

In this paper, we have argued that Algorithm Design – one of the key components of Computational Thinking – is *not* the same as *Algorithmic Reasoning*. In particular, we proposed a type of task schema that is crafted around the essential entity of algorithms and demonstrated how such a task schema can be uniformly applied to give rise to a sequence of tasks that brings the learner through the journey of Algorithmic Reasoning to advance to Creatively Founded Mathematical Reasoning. Crucially, this proposed task schema has been rigorously checked against the four CT+Math task design principles put forth in an earlier ATCM conference. Our present work can be seen as a further application of this set of design principles, thus demonstrating its versatility in the domain of task design that is specific to Computational Thinking used in Mathematics teaching and learning. The constraint of space in this paper has not allowed us the luxury of explaining how the activities described above can be carried out in an authentic classroom situation, what methodology we employ and how the desired positive results may be achieved with the students. Thus, one important future work would be to actualize what has been proposed in this paper.

References

[1] Bartlett, F. C. (1932). *Remembering: A study in experimental and social psychology.* Cambridge University Press.

[2] Boesen, J., Helenius, O., Lithner, J., Bergqvist, E., Bergqvist, T., and Palm, T. (2014). Developing mathematical competence: From the intended to the enacted curriculum, *Journal of Mathematical Behavior*, 33, pp. 72-87

[3] Brousseau, G. (1997). *Theory of didactical situations in mathematics.* Kluwer, Dordrecht

[4] Hiebert, J. (2003). What research says about the NCTM Standards. In J. Kilpatrick, G. Martin, & D. Schifter (Eds.), *A research companion to principles and standards for school mathematics*, National Council of Teachers of Mathematics, Reston, VA, pp. 5-26

[5] Haavold, P. (2011). What characterises high achieving students' mathematical reasoning? In B. Sriraman, & K. Lee (Eds.), *The elements of creativity and giftedness in mathematics* (Vol. 1) (pp. 193–215). SensePublishers.

[6] Ho, W. K., Looi, C. K., Huang, W., Seow, P. and Wu, L. (2019). Realizing Computational Thinking in Mathematics Classroom: Bridging the Theory-Practice Gap. In W.-C. Yang & D. Meades (Eds.), *Proceedings of the Twenty-fourth Asian Technology Conference in Mathematics*, Mathematics and Technology, LLC.

[7] Jonsson, B., Norqvist, M., Liljekvist, Y. and Lithner, J. (2014). Learning mathematics through algorithmic and creative reasoning. *The Journal of Mathematical Behaviour*, 36, pp. 20-32.

[8] Lester, F. K. and Kehle, P. (2003). From problem solving to modeling: The evolution of thinking about research on complex mathematical activity. In R. Lesh and H. M. Doerr (Eds.), *Beyond constructivism: Models and modeling perspectives on mathematics problem solving, learning, and teaching* (pp. 501-517). Mahwah, NJ: Erlbaum.

[9] Lithner, J. (2008). A research framework for creative and imitative reasoning. *Educational Studies in Mathematics, 67(3)*, 255–276.

[10] Lithner, J., Bergqvist, E., Bergqvist, T., Boesen, J., Palm. T, and Palmberg, B. (2010). Mathematical competencies: A research framework. In: *Mathematics and mathematics education: Cultural and social dimensions*. Bergsten, Jablonka & Wedege (Eds), Linköping, Sweden: Svensk förening för matematikdidaktisk forskning, SMDF, pp. 157-167

[11] Ministry of Education. (2019). *Mathematics Syllabuses: Secondary One to Four. Express Course, Normal (Academic) Course*. Curriculum Planning and Development Division, Singapore.

[12] Niss, M. (2003). Mathematical competencies and the learning of mathematics: The Danish KOM Project. In A. Gagatsis & S. Papastavridis (Eds.), *Proceedings of the Third Mediterranean Conference on Mathematics Education* (pp. 115–124). Athens, Hellenic Republic.

[13] Pólya, G. (1957). *How to Solve It*. Garden City, NY: Doubleday. p. 253.

[14] PRIMM website. Available at `https://primmportal.com/`.

[15] Raghubar, K. P., Barnes, M. A., and Hecht, S. A. (2010). Working memory and mathematics: A review of developmental, individual difference, and cognitive approaches. *Learning and Individual Differences, 20(2)*, 110–122.

[16] Papert, S. (1996). An Exploration in the Space of Mathematics Educations, *International Journal of Computers for Mathematical Learning*, **1**(1), 95 – 123.