K-8 Learning Trajectories Derived from Research Literature: Sequence, Repetition, Conditionals

Kathryn M. Rich
UChicago STEM Education
University of Chicago
kmrich@uchicago.edu

Carla Strickland
UChicago STEM Education
University of Chicago
castrickland@uchicago.edu

T. Andrew Binkowski
Department of Computer Science
University of Chicago
abinkowski@uchicago.edu

Cheryl Moran
UChicago STEM Education
University of Chicago
cgmoran@uchicago.edu

Diana Franklin
UChicago STEM Education
University of Chicago
dmfranklin@uchicago.edu

ABSTRACT

Computing curricula are being developed for elementary school classrooms, yet research evidence is scant for learning trajectories that drive curricular decisions about what topics should be addressed at each grade level, at what depth, and in what order. This study presents learning trajectories based on an in-depth review of over 100 scholarly articles in computer science education research. We present three levels of results. First, we present the characteristics of the 600+ learning goals and their research context that affected the learning trajectory creation process. Second, we describe our first three learning trajectories (Sequence, Repetition, and Conditionals), and the relationship between the learning goals and the resulting trajectories. Finally, we discuss the ways in which assumptions about the context (mathematics) and language (e.g., Scratch) directly influenced the trajectories.

CCS CONCEPTS

•Social and professional topics →Computational thinking; K-12 education;

KEYWORDS

K-6; Computational Thinking; Learning Trajectories

1 INTRODUCTION

Several school districts, including public schools in Chicago, New York City, and San Francisco, have begun CS for All initiatives that will bring computational thinking (CT) instruction to all students, elementary through high school. However, the knowledge to create research-based curricula - namely what, how, and when to teach CT material - is still emerging.

There exist two bodies of research literature to draw upon – one discussing what CT concepts students *should learn* [2, 22, 38] and the other exploring what students at different ages *did learn* through various CT-related activities [11, 12, 14, 16, 17, 20]. The wide breadth of literature presents a challenge: How do we extract cohesive learning trajectories for computational thinking concepts from a body of literature largely discussing individual, disconnected learning goals?

In this paper, we present the first three learning trajectories (Sequence, Repetition, and Conditionals) developed through analysis of over 100 published papers over the past decade. In particular, we make the following four contributions:

- Identify two study attributes common in current research literature that limit the studies' usefulness in creating full, empirically-supported learning trajectories.
- Articulate a decision-making process, grounded in education theory, to connect related learning goals.
- Present three learning trajectories and discuss their relationship to current literature.
- Discuss the ways in which assumptions about our context (integration with mathematics) and programming language (e.g., Scratch) directly influenced the trajectories.

The rest of the paper is organized as follows. We first present related work and our theoretical basis in Sections 2 and 3. We then describe our methods in Section 4. Section 5 analyzes attributes of the literature as it is relevant to this endeavor. Section 6 presents the learning trajectories, and Section 7 discusses the ways in which our assumptions influenced the trajectories. Finally, concluding thoughts are in Section 8.

2 BACKGROUND

Our project is based on prior work in computational thinking and is situated in particular curricular contexts.

2.1 Related Work

Wing's foundational paper describes computational thinking as "solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science," noting connections between CT and problem-solving strategies of other disciplines [38]. These connections have been further described by Barr and Stephenson [3] in a report on efforts to develop a definition of CT that is "coupled with examples that demonstrate how computational thinking can be incorporated in the classroom."

ICER'17, August 18-20, 2017, Tacoma, WA, USA.

DOI: http://dx.doi.org/10.1145/3105726.3106166

A challenge in bringing CS/CT to all has been understanding what topics, to what depth, and in what order computational thinking can be taught at different grades. Different tools and approaches have been used to explore this issue. Early work explored using LOGO to teach a variety of content areas at different ages [7, 8, 30]. The reemergence of computing in elementary schools has largely focused on Scratch, a popular block-based language. Seiter and Foreman [33] used existing Scratch projects across different grade levels to identify how computational thinking concepts varied by level in their Progression of Early Computational Thinking (PECT) Model. However, existence of a block in a project is not evidence of understanding the full meaning of the block, as Brennan et al. [5] found when interviewing Scratch contributors. Others have looked more directly at how CT content might be ordered. Dwyer et al. [11] explored lower anchor points and learning progressions for early work in algorithmic development, Franklin et al. [16] analyzed 4th, 5th, and 6th grade student work in sequence, events, and initialization, and Grover and Basu [20] explored student misconceptions of loops, variables, and Boolean logic.

2.2 Curricular Context

Learning trajectories influence curriculum, but they also are reciprocally influenced by curricular choices [9]. We made two curricular choices that influenced our trajectories.

Mathematics integrated with CT is a promising approach. Integration with a traditional subject connects new CT concepts to what students and teachers already know. Additionally, a large portion of the school day is already set aside for mathematics, allowing this new content to fit into an already-full K-5 curriculum, and there are many synergies between mathematics concepts and CT concepts.

Visual Block-Based Languages (VBBLs), such as Scratch [27], have become the language of choice for elementary level project-based curricula. The programming language and programming environment are the window through which learners interact with and learn about computational thinking. VBBLs are popular because they are used by dragging and dropping instructions (reducing syntax errors as well as typing, spelling, and memorization requirements) to create programs that involve the animation of 2-d images.

3 THEORETICAL BASIS

Our work was influenced by several bodies of work. First, we draw on the learning trajectory approach to defining and describing the components of a learning progression in a manner that makes it useful for guiding learning and instruction. Second, we draw on work reflecting a number of perspectives on how to organize and express the progression of topics at different ages (depicted four different ways in Figure 1).

3.1 Learning Trajectories

In our work, we use a learning trajectory (LT) approach. An LT is a map, or route, from knowledge that students bring to the classroom to more sophisticated and detailed understanding of a topic [9]. The starting point of the route is a crucial aspect of a trajectory; LTs are based on the assumption that their generation should be based on understanding of the current knowledge of the students being guided along the route [35] (a tenet of constructivism).

A learning trajectory begins as a thought experiment – a "hypothetical learning trajectory" in the words of Martin Simon [34] (p. 133) – that anticipates ways that students might be engaged in the kinds of thinking and learning necessary to move them toward the target understanding. These conjectures are then cyclically refined and tested in classrooms, with results with individual students or groups of students eventually being aggregated into a instructional sequence that might be considered "best-case" [9]. In this study, we attempted to synthesize scholarly literature to create initial hypothetical LTs to be tested in future work. A similar approach was used by Confrey, Maloney, and Corley [10], who developed learning trajectories by organizing and synthesizing the Common Core State Standards for Mathematics (CCSS-M), one of the most widely-adopted standards documents in U.S. education.

To operationalize the assembly of LTs from literature, we conceptualized LTs as having three parts: a set of learning goals, a developmentally appropriate path through those goals, and a set of illustrative activities that help students move along the path [9]. Our literature review process was designed with the aim of extracting information about these three elements from the articles reviewed. In this paper, we focus on the first two elements of learning trajectories: goals and pathways. Thorough discussion of tasks is beyond the scope of this paper.

3.2 Shapes of Learning Trajectories

Several theories of learning influenced the manner in which we organized our trajectories. The influence of each of these theories is described in Section 4.

First, learning progressions have been proposed that illustrate a path (or choice of paths) that a student goes through while developing understanding of a particular topic, with the final goal being some level of understanding appropriate to the student's age and experience [4]. This might lead to a trajectory with a shape as depicted in Figure 1a - largely linear with some choices in path.

Other scholars have made the argument that learning progression models, in general, are too simple to capture the learning process. Hammer and Sikorski [23] argue that a particular level of understanding requires different pieces of knowledge. These pieces of knowledge may not have a particular order and may be activated differently depending on the learning context. This perspective might lead to a trajectory more like Figure 1b, with many more independent pieces of knowledge that, once learned, result in a deeper or more complete understanding.

A third influence on the shape of our trajectories is theory inspiring spiral curricula, itself influenced by cognitive theory advanced by Jerome Bruner [6]. Bruner wrote, "We begin with the hypothesis that any subject can be taught in some intellectually honest form to any child at any stage of development." That is, complex topics can be simplified in various ways for young children. Those same topics are revisited multiple times, teaching more depth and complexity as students mature. In this case, the shape of a learning trajectory would look like Figure 1c, in which students revisit some of the same topics each year, building some new knowledge and complexity each time.

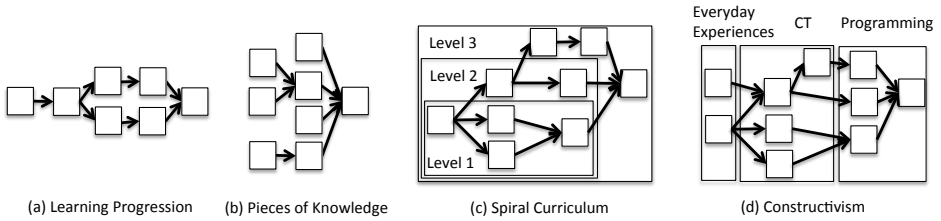

Figure 1: Shapes and Content of LTs as influenced by theoretical framings.

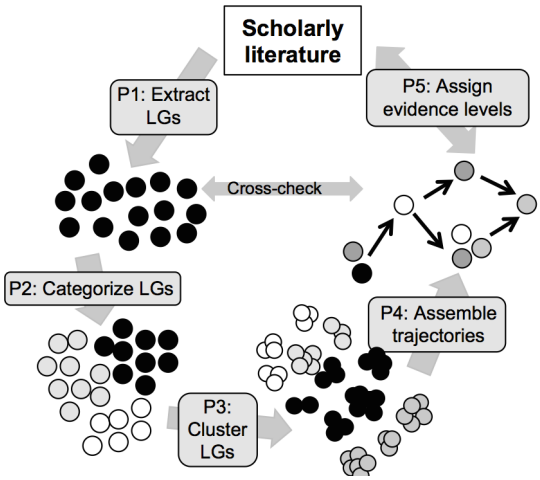

Figure 2: The learning trajectories creation process

Finally, our work is influenced by the constructivist assumption that all effective learning utilizes knowledge that learners already possess [34]. This perspective might lead to a trajectory that begins with everyday experiences, then "filters" them to identify the aspects that are most relevant to the new learning, gradually refining everyday knowledge to apply to academic topics of interest. For our work, we conceptualize the progression as starting with everyday experiences, proceeding to broad CT ideas, then applying those ideas to programming, as shown in Figure 1d.

4 METHODS

Our methods for building learning trajectories take place in five iterative phases: (1) extracting learning goals (LGs) from literature, (2) categorizing learning goals, (3) clustering learning goals, (4) assembling clusters into trajectories, and (5) assigning evidence levels to goals and relationships. We describe each of these phases in the sections that follow. The overall process is summarized in Figure 2.

Phase 1: Extracting Learning Goals from Literature

The literature search included all articles that cited Wing's [38] paper on computational thinking, keyword searches of the Educational Research Information Center (ERIC) database, and proceedings from the 2006–2016 Special Interest Group on Computer Science Education (SIGCSE) Technical Symposium and the Innovation and Technology in Computer Science Education (ITiCSE). Application of retention criteria described in [31] resulted in 108 articles.

Table 1: Concept categories for sorting learning goals

Concepts[21]
Abstraction and pattern generalization
Systematic processing of information
Symbol systems and representations
Algorithmic notions of flow of control
Structured problem decomposition
Iterative, recursive, and parallel thinking
Conditional logic
Efficiency and performance constraints
Debugging and systematic error detection

We define a learning goal as *any explicit statement or implicit endorsement of what students can or should be able to do in relation to computational thinking.* Example heuristics for identifying learning goals that meet this definition can be found in [31]. When possible, text was recorded verbatim from the article, aside from minor edits to be understood out of context. This process resulted in 671 learning goals.

Phase 2: Categorizing Learning Goals

As each learning goal was extracted from an article, it was categorized by concept and support type. Concepts, taken from [21], are listed in Table 1. The two support types are *student* and *theoretical*.

Student support: The authors describe evidence that students of particular ages were able to engage with the learning goal.

Theoretical support: The authors describe the goal and/or its appropriateness for students without citing any particular evidence other than their own expertise or work with learners outside the target student population (e.g., teachers or other adults).

Each article was independently reviewed by at least two researchers to extract and categorize learning goals. If intercoder agreement of at least 80% could not be reached through discussion, a third reader coded the LGs from the article independently and decided the remaining discrepancies.

Phase 3: Clustering Learning Goals

Learning goals were sorted into large groups according to combinations of concepts. In this paper, we discuss the set of LGs tagged as related to Conditional logic, Algorithmic notions of flow of control, or Iterative, recursive, and parallel thinking. We chose these particular concepts because they were widely discussed in the literature and had the most concrete learning goals, making them a useful starting point for refining our process. In future work, we plan to develop learning trajectories for the other aspects of CT articulated in Table 1.

Four researchers independently sorted this collection of LGs into categories, with no predetermined categories. The researchers then met to discuss what major themes emerged. Based on these thematic discussions, new categories were defined, and researchers independently re-sorted the original collection into these categories. Any LG that was sorted into the same category by at least three of the four researchers was retained, with any goals that did not meet this criteria removed from the collection. The resulting collections of goals were called clusters. This paper discussed the LTs created from three clusters: Sequence, Repetition, and Conditionals.

Phase 4: Assembling Clusters into Trajectories

We used a two-stage process to assemble each cluster of learning goals into a trajectory. Due to the limited information available within the literature (as discussed in Section 5), we gathered an interdisciplinary team consisting of experts in computer science, computational science, computer science education, mathematics education, and mathematics curriculum to agree upon decisions involving the trajectories.

First, we wrote "consensus goals" (CGs) to articulate big ideas expressed within a cluster and assigned to each CG the learning goals that supported it. An LG was characterized as directly supporting a CG if the CG could be considered a restatement of the LG. An LG was considered inferred support for the CG if it was not a direct restatement, but was more specific than the CG, represented an idea for which the CG was a prerequisite, or lacked sufficient detail to provide direct support.

Second, we attempted to articulate relationships and dependencies among the consensus goals that define potential pathways through the goals. To do so, we applied the theories discussed in Section 3.2 as follows. To apply the theory of learning progressions, we noted any study results that suggested a progression in difficulty (e.g., when students could complete one aspect of a task but not another), and used this information to place the less difficult ideas before the more difficult ideas.

We then adopted three general organizational heuristics based on the pieces-of-knowledge theory, spiral curriculum ideas, and constructivist assumptions, respectively:

1. Address and develop component ideas separately before expecting them to be applied in concert.

2. Identify the minimum knowledge needed to create an artifact applying a concept, and place this knowledge into a progression. Then add to the trajectory by adding alternative, branching paths that add layers of complexity to the use of the concept.

3. Begin with the ideas most closely connected to students' everyday lives, continue toward broad CT concepts, and finally apply CT to programming. In practice, this resulted in a pattern of addressing "unplugged" ideas before programming-specific ideas.

In addition to these heuristics, we also applied knowledge of mathematical skills, pedagogical approaches used in mathematics instruction, and likely programming languages that would be used in age-appropriate activities. We discuss the ways in which these assumptions influenced the resulting trajectories in Section 7.

Phase 5: Assigning Evidence Levels

A last cross-check was performed on all consensus goals with relevant learning goals to ensure that the CGs aligned with the original intent of the LGs.

The collection of learning goals supporting each consensus goal was examined. If at least one learning goal supporting a CG had student support, the CG was marked with student support; otherwise, the CG was marked with theoretical support. Any CGs without learning goals supporting them were removed. We also identified which arrows were supported by the literature and which were a result of professional judgement guided by the heuristics above.

5 LITERATURE ATTRIBUTES

In this section, we describe two attributes of the CS education literature that made it difficult to create empirically-supported learning trajectories.

Little research in K-6 has been performed on students deliberately chosen to have computing experience.

Only 14% of the studies that produced LGs with student support stated that the students had some programming/CT experience. The rest either stated that students had mixed experience (10%), unclear experience (43%), or no experience (33%). As a result of the low experience level of most study participants, similar goals have been attempted by students of disparate ages. For example, one study showed that eighth and ninth graders could decompose a game into a sequence of frames [32], while another showed that Kindergarteners can parse a sequence of events into component steps [14]. These goals could be generalized into the same consensus goal about parsing a large task into parts, but the wide range of grade levels gives little information about where this goal should be placed within an age-graded trajectory. In addition, the grade range in a study was sometimes too wide to be useful, such as an analysis of Scratch projects created by students aged 8–18 in an after-school program [28].

As a result, we were largely unable to use the ages or grade levels of students to inform the relationships between consensus goals. We instead relied heavily on our theoretical underpinnings to determine relationships among goals.

Most research in K-8 focuses on independent learning goals rather than the dependency between multiple learning goals

A related attribute of the literature is that most research (81%) focused on a single learning goal or independent learning goals rather than the relationship between multiple learning goals. For example, studies asked questions such as, did students learn how to think in parallel [19]? Or, are students able to create programs with Booleans, variables, conditionals, loops, functions, or events, which were the programming constructs taught [25]? This is related to the fact that most studies were on novice learners – most interventions were short, so students would not learn enough in a single intervention to study the relationships between different learning goals. This was another factor that led us to rely on educational theories to determine relationships among consensus goals.

6 LEARNING TRAJECTORIES

We now present three learning trajectories: Sequence, Repetition, and Conditionals. We begin by presenting numerical summaries of the trajectories, shown in Tables 2 and 3. Table 2 shows the number of learning goals in each cluster, detailing how many had student support and how many had theoretical support. In the

Table 2: Learning goals for each cluster

Trajectory	Total LGs (Student / Theoretical)	LGs used in trajectory
Sequence	48 (34/14)	37
Repetition	47 (30/17)	42
Conditionals	49 (34/15)	49

Table 3: CGs and arrows for each trajectory

Trajectory	Total CGs (Offline / Computer-based)	Total arrows	Literature-supported arrows
Sequence	10 (5/5)	11	3
Repetition	10 (5/5)	13	0
Conditionals	12 (6/6)	18	3

Table 4: Three LGs leading to one Sequence CG

LG1	Correctly sequence a set of computational instructions. [13]
LG2	"[P]ut instructions in the correct sequence". [1]
LG3	"[G]iven lines of pseudocode that need to be put in the correct order to solve a problem", put them in order. [24]
CG	The order in which instructions are carried out can affect the outcome.

cases of Sequence and Repetition, a handful of learning goals were discarded during the trajectory construction process due to being miscategorized or being identified as the same goal extracted twice from different reports of the same study. The number of LGs used to support the final trajectory is shown in the third column. All the LGs in the Conditionals cluster were used in the trajectory.

Table 3 shows the number of consensus goals and arrows in each trajectory, along with how many of the CGs are offline goals versus computer-based goals, and how many of the arrows were supported by information extracted from the literature.

Pictorial summaries of the trajectories are shown in Figures 3, 4, and 5. Gray boxes indicate offline, or "unplugged," CGs, while white boxes indicate computer-based CGs. Arrows indicate two types of relationships between consensus goals. First, a black arrow indicates that the understanding of the source box is necessary (or at least very helpful) to understand the destination box. Second, a grey arrow indicates that traversing through the previous box is *helpful*, but not always *necessary* to reach the destination box. These grey arrows are the beginnings of a spiral curriculum, allowing students to make multiple paths through the same material, gaining more understanding each time. Finally, information on the lower right-hand corner of each box indicates the number of LGs that were coalesced into this consensus goal and the stronger type of evidence found for a CG or relationship arrow (*S* for student support or *T* for theoretical support).

We now discuss each trajectory in more depth with details on the LT construction process, including an example of a set of learning goals that were coalesced into a single consensus goal, and a discussion of the ideas summarized in the trajectory.

6.1 Sequence

Construction Process The Sequence cluster included 48 learning goals, 34 with student support and 14 with theoretical support. Table 4 lists the three learning goals that were coalesced into the single "The order in which instructions are carried out can affect the outcome." consensus goal. This example shows two things. First, the three learning goals were nearly identical, discussing various activities involving putting instructions in order. Second, the consensus goal was expressed as an *understanding* goal rather than an *action* goal. To articulate the consensus goal, we determined a core knowledge students would either learn from or demonstrate their understanding of by performing the specified actions.

As the learning goals were synthesized into consensus goals, two big ideas emerged: Precision and Order, as seen in the two branches in Figure 3. Following Heuristic (1) discussed in Phase 4 of the Methods section, we placed CGs relating to each big idea separately before CGs that combined them. Following Heuristic (3), we articulated a path through each big idea that started with everyday knowledge, proceeded to CT, and later addressed programming. During this process, we also made use of three student-supported dependence relationships pulled from the literature.

In this trajectory, both the early, offline CT goals and the later programming goals are well-supported by LGs with student support. This was not the case for the other two trajectories, as discussed below.

Trajectory Description The Sequence trajectory begins with two branches: Precision and Order. The beginning treatment includes the Precision branch, whereas the intermediate treatment includes both branches and the coordination of the two ideas. Note that the arrows between consensus goals within the Precision branch are largely black, while the arrows within the Order branch are largely grey. This means that the Order branch shows a possible path, not a path based on strict dependencies. The order of CGs within the Order branch, as well as the choice to put this branch in the intermediate treatment instead of the beginning treatment, was influenced by our work in mathematics, as described in Section 7.

After the branches meet, our two subsequent consensus goals are springboards to other learning trajectories. "Some commands modify the default order of execution, altering when and which instructions are executed." leads to Repetition and Conditionals, described below. "The position of a new command can affect outcomes." describes the modification process, as opposed to creation process, of programming. This not only relates to understanding instructions and ordering, but it also relates to debugging techniques, the subject of a future learning trajectory not included in this paper. Due to the need for coordination with other topics, we place these CGs in the advanced treatment of Sequence.

6.2 Repetition

Construction Process The Repetition (iteration / loop) trajectory was assembled from 47 LGs, with a 30/17 split on student support vs. theoretical support. Table 5 lists the four LGs that were used to create one CG. This example illustrates the ways in which we inferred support from seemingly disparate LGs into one CG. LG1 articulates an understanding that restating instructions in fewer commands can be desirable, implying that children should know

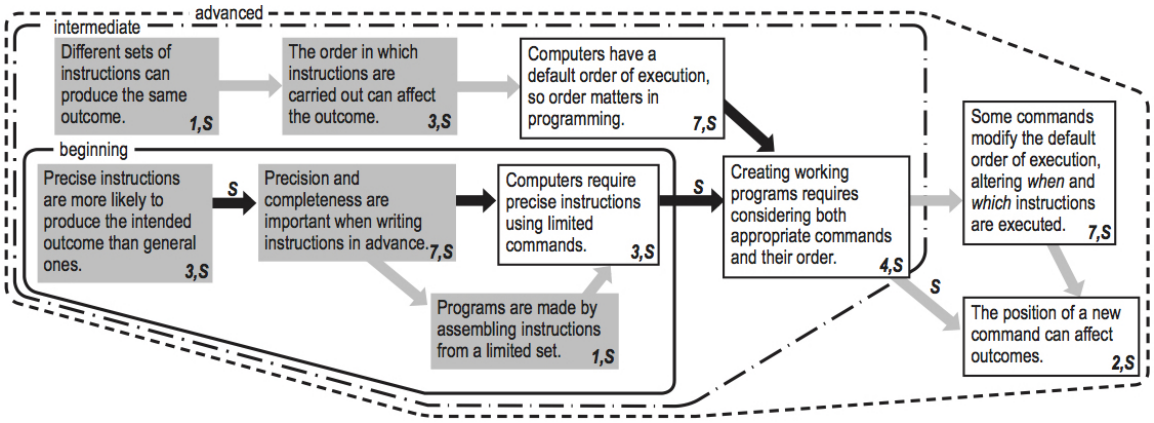

Figure 3: Sequence learning trajectory.

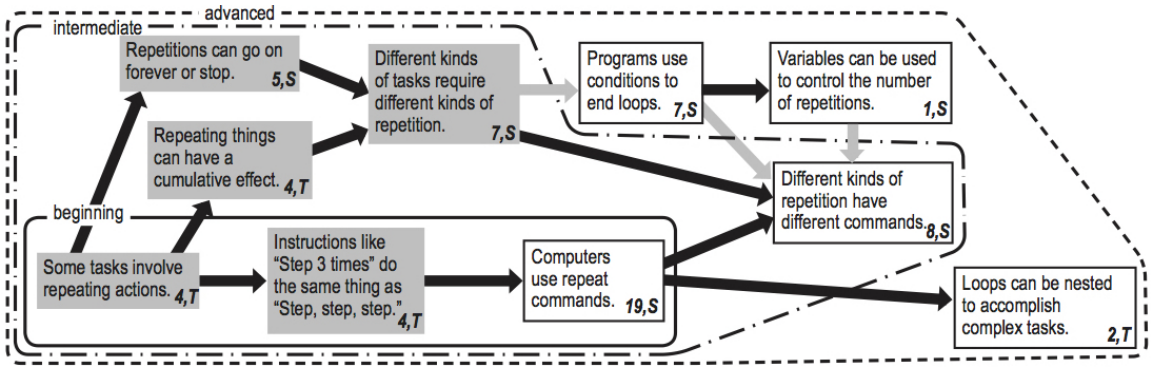

Figure 4: Repetition learning trajectory.

Table 5: Four LGs leading to one Repetition CG

LG1	Recognize when it is not desirable to "repeat code in the same project". [29]
LG2	Identify "loop constructs but not necessarily working code." [18]
LG3	Use "explicit looping". [22]
LG4	Use "loops" (as an example of abstraction). [3]
CG	Instructions like "Step 3 times" do the same thing as "Step, step, step."

how to do so. LG2 points out that students can be expected to think about a loop construct without using it in a program, implying that a broader, offline CT goal can appear before a programming goal. "Explicit looping" in LG3 refers to using repetition and being aware that you are doing so, implying that you should be able to express the repetition ideas explicitly. Finally, LG4 points out that using a loop requires abstraction, which we specify as identifying what repeats and how many times it should repeat. Thus we consider all of these LGs to be inferred support for the understanding of how to write instructions using repetition language.

Following Heuristic (3), the first CG in this trajectory relates to how repetition manifests in everyday life. The LT then proceeds towards broad CT ideas related to repetition and finally proceeds to programming goals. Interestingly, we found that though approximately 64% of the LGs in this cluster had student support, these tended to support the five programming consensus goals – in particular, the CG about basic understanding and use of a loop command ("Computers use repeat commands."). We relied heavily on theoretical literature to articulate the early CGs, and defined the pathways based entirely on our heuristics.

Trajectory Description Figure 4 depicts the resulting Repetition learning trajectory. It begins with the idea that repetition is used for many tasks, then expands into three branches addressing broad CT ideas related to repetition. The first relates to expressing the need to repeat instructions via a simple, countable loop – this is what we suggest for beginners. The other two branches relate to recognizing the power of repetition ("Repeating things can have a cumulative effect.") and how and when to stop a repetition. At the end of the intermediate treatment, students have an understanding of how and when to use multiple kinds of loops. An advanced treatment introduces nested loops and using variables (and conditions) to control loops.

6.3 Conditionals

Construction Process The Conditionals trajectory was assembled from 49 LGs, with a 34/15 split on student support vs. theoretical

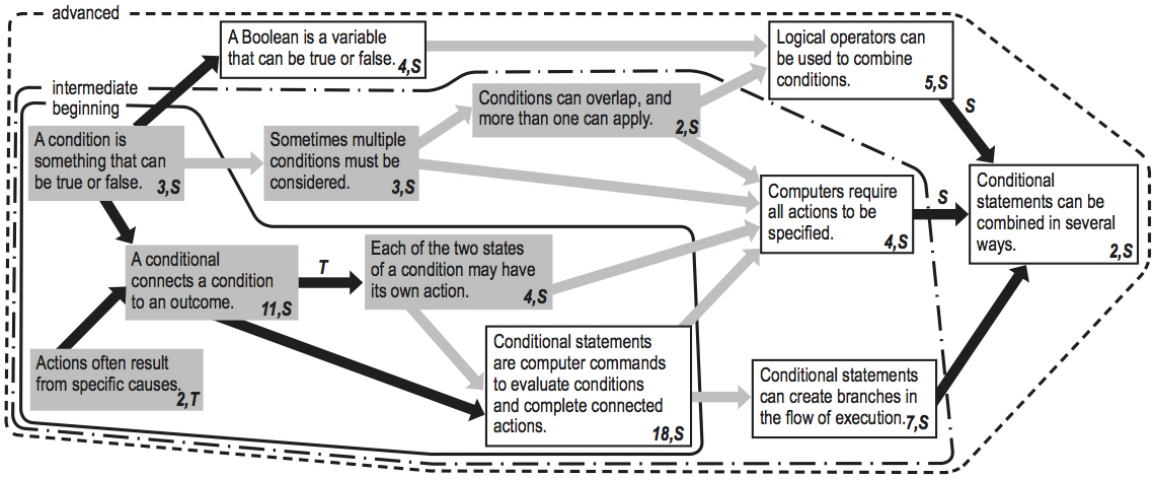

Figure 5: Conditionals learning trajectory.

Table 6: Four LGs leading to one Conditionals CG

LG1	Interpret the results of conditional statements [26]
LG2	Use an "if/else statement" [37]
LG3	Use an if/else statement [33]
LG4	"Write an algorithmic solution ... that takes into account all boundary conditions" [2]
CG	Each of the two conditional states may have its own action.

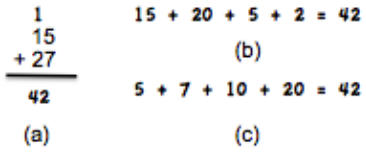

Figure 6: Three ways to add 15 + 27. (a) uses a rote process, whereas (b) and (c) show different orders and steps.

support. Table 6 lists the four LGs that were coalesced into "Each of the two states of a condition may have its own action." This example illustrates that the context of the trajectory is necessary to see the relationship between the LGs and the resulting CG. On its face, the LGs do not necessarily speak directly to the importance of considering the false state. However, this is what distinguishes this CG from its predecessors – other LGs referred only to an if statement or generally to conditionals. The presence of the else and the stated desire to take into account all boundary conditions resulted in a separate CG focused on paying special attention to considering the false state of a condition and specifying the else clause of a conditional.

Following Heuristic (3), we began the Conditionals LT with the basic understanding of cause and effect and of evaluating the truthfulness of a statement. Spiral curriculum ideas (Heuristic (2)) strongly influenced the shape of this trajectory, as analysis of the LGs in this cluster revealed many layers of complexity both in consideration of possible conditions and in how conditionals might be implemented.

Trajectory Description The Conditionals trajectory is spiraled to present three levels of advancement in understanding. At a beginning level, students learn the binary nature of conditions, the cause-and-effect connection between condition states and outcomes, and the commands necessary to use conditionals in a program. At this level of understanding, event-based programs could be used without explicit if-then statements. An intermediate treatment adds multiple conditions and overlapping conditions, requiring multiple or nested conditionals, as well as explicit if-then-else code that breaks the sequential execution of a single code snippet. Finally, an advanced treatment introduces Boolean variables.

7 DISCUSSION

In the absence of evidence from the literature to order consensus goals, or when there were multiple reasonable possibilities, two extra pieces of information were used to influence the ordering: mathematics and programming language. In this section, we discuss more thoroughly specific examples of how mathematics and programming language assumptions influenced the decisions made.

7.1 Mathematics

Mathematics influenced our trajectories by providing an existing order in which CT concepts related to mathematics were introduced. In essence, where the CT literature was lacking in tying concepts to grade levels, mathematics instruction provided insight into existing expectations for students to use those concepts, even if they are not presented as such.

For example, in the Sequence LT presented in Section 6.1, which concept should be taught first – that different instructions can have the same outcome or that the order of instruction affects the outcome? That is, should understanding *instructions* come first or the effect of *order*? If we look to the CS K-12 Framework [15], we see that order comes first.

Research-supported pedagogical approaches to early exploration of mathematical operations, however, de-emphasize the importance of order of steps. A traditional approach to teaching two-digit addition is depicted in Figure 6a. To add 15 + 27, the digits in the

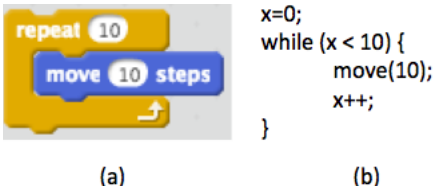

Figure 7: 10-iteration loop, (a) Scratch and (b) C.

Figure 8: Testing a pixel for black, (a) Scratch and (b) C.

ones places are added (5 + 7 = 12). We write the 2 in the ones column of the sum and carry the 1. Then 1 + 1 + 2 are added, resulting in 4. We write the 4 in the tens column to get the final answer of 42. This is a specific algorithm in which the order of the steps matters. However, another approach encourages students to explore the meaning of addition and invent their own methods for computation. As they gain understanding of the counting sequence, children often invent "jump" strategies such as jumping up 20 from 15, then jumping up 5 more, then 2 more, as shown in Figure 6b [36]. A different student may prefer to start from 0, then jump up by the ones (5 from 15 and 7 from 27) and then by the tens (10 from 15 and 20 from 27), as shown in Figure 6c. Both methods result in the same answer, 42. The additions could be performed in any order, and the numbers broken up in more than one way.

Now looking closely at the Order branch of the Sequence trajectory, we see that it reflects the order of math instruction, placing "Different sets of instructions can produce the same outcome." before "The order in which instructions are carried out can affect the outcome." Instead of the first puzzles in the code.org curriculum, in which students use forward and turn blocks to navigate a character through a maze with obstacles blocking all alternate paths, students would be given a project that is explicitly designed to allow multiple solutions. We do not have empirical evidence that would tell us which approach is better. Instead, we observe that the trajectory depicted in this paper is likely to align better with mathematics.

7.2 Programming Language

Finally, we found that assumptions about the programming language used in elementary school influenced the dependencies for the Repetition and Conditionals trajectories. In many languages used by college students and industry professionals, loops and conditionals are much more complex than in Scratch. In contrast, Scratch is specifically designed to provide "low floors and high ceilings." This means that while Scratch includes very sophisticated instructions that require knowledge of percentages, negative numbers, and variables, it also includes blocks with minimal requirements.

In Scratch, a programmer can set up a loop to execute 10 times by merely entering 10 as an argument into a repeat block. An equivalent loop in most languages, as shown in Figure 7, would require the use of variables and conditionals, both more advanced concepts than a simple repeat block. This means that to use a loop in a Scratch-like language, students need only be able to count – a Kindergarten skill. This is reflected in the beginning level of the Repetition trajectory. All that is necessary is to understand that "Step 3 times" is the same as "Step, step, step." This is the only prerequisite knowledge to using a simple repeat command; there are no connections between Conditionals and Repetition in early levels.

Likewise, a meaningful use of a conditional in conventional text-based languages requires variables and a comparison operator, as shown in Figure 8. Scratch, on the other hand, provides condition primitives such as "touching [other sprite]" or "touching [color]" in order to allow one to create code that reacts to conditions in the visual screen. This affects the Conditionals trajectory because students can use conditionals in the elementary level, yet Boolean variables can be delayed until the advanced level.

8 CONCLUSIONS

This paper presents detailed learning goals, and connections between them, for three CT concepts: Sequence, Repetition, and Conditionals. We provide insight into how these topics can be developed by harnessing knowledge from students' everyday experiences, potentially leading to broader understanding of how CT applies to the world than would result from teaching through programming alone. We also discuss how integration context and programming languages can influence (though not determine) learning trajectories.

This is merely a starting point. More empirical data is necessary to better understand the relative difficulty of different concepts, how they map to specific grade levels, and how they change based on integration with specific subjects. Finally, there are several more trajectories to be created, such as Debugging, Decomposition, and Abstraction.

ACKNOWLEDGMENTS

This material is based on work supported by the National Science Foundation under Award 1542828 and Award 1240985. Any opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect those of the National Science Foundation.

REFERENCES

[1] Charoula Angeli, Joke Voogt, Andrew Fluck, Mary Webb, Margaret Cox, Joyce Malyn-Smith, and Jason Zagami. 2016. A K-6 computational thinking curriculum framework: implications for teacher knowledge. *Educational Technology & Society* 19, 3 (2016), 47–58.

[2] Michal Armoni and Judith Gal-Ezer. 2014. Early Computing Education: Why? What? When? Who? *ACM Inroads* 5, 4 (Dec. 2014), 54–59. DOI: http://dx.doi.org/10.1145/2684721.2684734

[3] Valerie Barr and Chris Stephenson. 2011. Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community? *ACM Inroads* 2, 1 (Feb. 2011), 48–54. DOI: http://dx.doi.org/10.1145/1929887.1929905

[4] Michael T Battista. 2011. Conceptualizations and issues related to learning progressions, learning trajectories, and levels of sophistication. *The Mathematics Enthusiast* 8, 3 (2011), 507–570.

[5] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In *In AERA 2012*.

[6] Jerome Bruner. 1960. *The Process of Education*. The President and Fellows of Harvard College, Cambridge, MA, USA.

[7] Douglas H. Clements. 2002. Computers in Early Childhood Mathematics. *Contemporary Issues in Early Childhood* 3, 2 (2002), 160–181.

[8] Douglas H. Clements and J Sarama. 1997. Logo: A Decade of Progress. *Computers in Schools* 14, 1/2 (1997), 9–46.

[9] Douglas H. Clements and J Sarama. 2004. Learning Trajectories in Mathematics Education. *Mathematical Thinking and Learning* 6, 2 (2004), 81–89.

[10] Jere Confrey, Alan P. Maloney, and Andrew K. Corley. 2014. Learning trajectories: a framework for connecting standards with curriculum. *ZDM* 46, 5 (2014), 719–733. DOI: http://dx.doi.org/10.1007/s11858-014-0598-7

[11] Hilary Dwyer, Charlotte Hill, Stacey Carpenter, Danielle Harlow, and Diana Franklin. 2014. Identifying Elementary Students' Pre-instructional Ability to Develop Algorithms and Step-by-step Instructions. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. ACM, New York, NY, USA, 511–516. DOI: http://dx.doi.org/10.1145/2538862.2538905

[12] G Fessakis, E Gouli, and E Mavroudi. 2013. Problem Solving by 5-6 Years Old Kindergarten Children in a Computer Programming Environment: A Case Study. *Computers Education* 63 (April 2013), 87–97. https://www.learntechlib.org/p/132276

[13] Louise P Flannery and Marina Umaschi Bers. 2013. Letfis dance the firobot hokey-pokey!fi Childrenfis programming approaches and achievement throughout early cognitive development. *Journal of research on technology in education* 46, 1 (2013), 81–101.

[14] Louise P. Flannery, Brian Silverman, Elizabeth R. Kazakoff, Marina Umaschi Bers, Paula Bontá, and Mitchel Resnick. 2013. Designing ScratchJr: Support for Early Childhood Learning Through Computer Programming. In *Proceedings of the 12th International Conference on Interaction Design and Children (IDC '13)*. ACM, New York, NY, USA, 1–10. DOI: http://dx.doi.org/10.1145/2485760.2485785

[15] K-12 Computer Science Framework. 2016. K-12 Computer Science Framework. (2016). https://k12cs.org/

[16] Diana Franklin, Gabriela Skifstad, Reiny Rolock, Isha Mehrotra, Valerie Ding, Alexandria Hansen, David Weintrop, and Danielle Harlow. 2017. Using Upper-Elementary Student Performance to Understand Conceptual Sequencing in a Blocks-based Curriculum. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. ACM, New York, NY, USA, 231–236. DOI: http://dx.doi.org/10.1145/3017680.3017760

[17] Michelle Friend and Robert Cutler. 2013. Efficient Egg Drop Contests: How Middle School Girls Think About Algorithmic Efficiency. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research (ICER '13)*. ACM, New York, NY, USA, 99–106. DOI: http://dx.doi.org/10.1145/2493394.2493413

[18] Ursula Fuller, Colin G Johnson, Tuukka Ahoniemi, Diana Cukierman, Isidoro Hernán-Losada, Jana Jackova, Essi Lahtinen, Tracy L Lewis, Donna McGee Thompson, Charles Riedesel, and others. 2007. Developing a computer science-specific learning taxonomy. In *ACM SIGCSE Bulletin*, Vol. 39. ACM, 152–170.

[19] Chris Gregg, Luther Tychonievich, James Cohoon, and Kim Hazelwood. 2012. EcoSim: a language and experience teaching parallel programming in elementary school. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, 51–56.

[20] Shuchi Grover and Satabdi Basu. 2017. Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. ACM, New York, NY, USA, 267–272. DOI: http://dx.doi.org/10.1145/3017680.3017723

[21] Shuchi Grover and Roy Pea. 2013. Computational thinking in K–12: A review of the state of the field. *Educational Researcher* 42, 1 (2013), 38–43.

[22] Mark Guzdial. 2008. Education: Paving the Way for Computational Thinking. *Commun. ACM* 51, 8 (Aug. 2008), 25–27. DOI: http://dx.doi.org/10.1145/1378704.1378713

[23] D. Hammer and T. Sikorski. 2015. Implications of complexity for research on learning progressions. *Science Education* 99, 3 (2015), 424–431.

[24] Celine Latulipe, N Bruce Long, and Carlos E Seminario. 2015. Structuring flipped classes with lightweight teams and gamification. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. ACM, 392–397.

[25] Michael J Lee, Faezeh Bahmani, Irwin Kwan, Jilian LaFerte, Polina Charters, Amber Horvath, Fanny Luor, Jill Cao, Catherine Law, Michael Beswetherick, and others. 2014. Principles of a debugging-first puzzle game for computing education. In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on*. IEEE, 57–64.

[26] Colleen M Lewis. 2010. How programming environment shapes perception, learning and goals: logo vs. scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education*. ACM, 346–350.

[27] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. *Trans. Comput. Educ.* 10, 4, Article 16 (Nov. 2010), 15 pages. DOI: http://dx.doi.org/10.1145/1868358.1868363

[28] John H. Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. 2008. Programming by Choice: Urban Youth Learning Programming with Scratch. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '08)*. ACM, New York, NY, USA, 367–371. DOI: http://dx.doi.org/10.1145/1352135.1352260

[29] Jesús Moreno and Gregorio Robles. 2014. Automatic detection of bad programming habits in scratch: A preliminary study. In *Frontiers in Education Conference (FIE), 2014 IEEE*. IEEE, 1–4.

[30] Seymour Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA.

[31] Kathryn Rich, Carla Strickland, and Diana Franklin. 2017. A Literature Review through the Lens of Computer Science Learning Goals Theorized and Explored in Research. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 495–500.

[32] Emmanuel Schanzer, Kathi Fisler, Shriram Krishnamurthi, and Matthias Felleisen. 2015. Transferring Skills at Solving Word Problems from Computing to Algebra Through Bootstrap. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. ACM, New York, NY, USA, 616–621. DOI: http://dx.doi.org/10.1145/2676723.2677238

[33] Linda Seiter and Brendan Foreman. 2013. Modeling the Learning Progressions of Computational Thinking of Primary Grade Students. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research (ICER '13)*. ACM, New York, NY, USA, 59–66. DOI: http://dx.doi.org/10.1145/2493394.2493403

[34] M. A. Simon. 1995. Reconstructing mathematics pedagogy from a constructivist perspective. *Journal for Research in Mathematics Education* (1995), 114–145.

[35] Martin A Simon and Ron Tzur. 2004. Explicating the role of mathematical tasks in conceptual learning: An elaboration of the hypothetical learning trajectory. *Mathematical thinking and learning* 6, 2 (2004), 91–104.

[36] Lieven Verschaffel, Brian Greer, and Erik De Corte. 2007. Whole number concepts and operations. In *Second handbook of research on mathematics teaching and learning*. Information Age Publishing, 557–628.

[37] Linda Werner, Jill Denner, and Shannon Campe. 2015. Children programming games: a strategy for measuring computational learning. *ACM Transactions on Computing Education (TOCE)* 14, 4 (2015), 24.

[38] Jeannette M. Wing. 2006. Computational Thinking. *Commun. ACM* 49, 3 (March 2006), 33–35. DOI: http://dx.doi.org/10.1145/1118178.1118215