Decomposition: A K-8 Computational Thinking Learning Trajectory

Kathryn M. Rich
Michigan State University
richkat3@msu.edu

Carla Strickland
UChicago STEM Education
castrickland@uchicago.edu

T. Andrew Binkowski
University of Chicago
abinkowski@uchicago.edu

Diana Franklin
UChicago STEM Education
dmfranklin@uchicago.edu

ABSTRACT

As new initiatives in computational thinking and computer science (CS/CT) are being developed and deployed, it is important to identify and understand the key concepts that are essential for student learning. In this study, we present the phases of construction of a learning trajectory (LT) for Decomposition in the context of CS/CT in K-8 education. From an extensive literature review, 63 learning goals representative of decomposition understanding and practices were identified and synthesized into 13 consensus goals. The focus of this paper is how relationships between these consensus goals were identified and used to place the goals into a learning trajectory. We discuss the theories and frameworks that guided the trajectory's construction as well as the methodology and justifications used to draw pathways through the trajectory in each phase. Finally, we discuss potential uses for the trajectory and suggest further explorations for decomposition in CS/CT.

CCS CONCEPTS

• **Social and professional topics** → **Computational thinking**;
K-12 education; • **Applied computing** → **Education**;

KEYWORDS

Decomposition, Computational thinking, K-8

ACM Reference Format:

Kathryn M. Rich, T. Andrew Binkowski, Carla Strickland, and Diana Franklin. 2018. Decomposition: A K-8 Computational Thinking Learning Trajectory. In *ICER '18: 2018 International Computing Education Research Conference, August 13–15, 2018, Espoo, Finland*. ACM, New York, NY, USA, Article 4, 9 pages. <https://doi.org/10.1145/3230977.3230979>

1 INTRODUCTION

Decomposition is a fundamental skill in problem solving, as highlighted by the modularity subconcept in the programs and algorithms concept of both the K-12 CS Framework [13] and the CSTA K-12 CS Standards [3]. While these resources provide guidance

on what students can understand or do at the end of grade bands, learning trajectories seek more detailed information about how such skills could be developed. There are many expected practices of decomposition in CS/CT, for example, breaking a large block of code into functional units [3, 13] or designing a game level-by-level [20, 34]. However, little is known about how these skills might develop incrementally, building from students' everyday experiences.

In this light, we ask the research question, "How can learning goals about decomposition expressed in separate research papers be used to construct a learning trajectory that will be useful to researchers and practitioners?" We present a learning trajectory (LT) for Decomposition and the methodology and approaches for its creation. In particular, we make the following contributions:

- Summarize the explicit and implicit ways that existing research sheds light on students' development of decomposition practices.
- Present an LT for Decomposition, explicating how we used existing research and educational theory across its several stages of development.
- Discuss several ways in which the completed LT might be used by researchers, curriculum developers, and practitioners to advance efforts in K-8 computer science education.

The rest of this paper is organized as follows. We begin by summarizing how decomposition has been studied in CS/CT education literature and presenting the theoretical frames that guided our work. We then present our synthesis of more than 60 independent decomposition-related learning goals articulated by other researchers into a representative set consensus goals. Next, we describe the construction of the Decomposition learning trajectory in four phases guided by different criteria: literature support, constructivist learning theory, computer science teaching pedagogy, and higher-level analysis of dimensions that emerged. Finally, we close with a discussion the various ways we believe the Decomposition trajectory can be used in future research and practice.

2 BACKGROUND

In this section, we review related work on decomposition and on learning trajectories. We also introduce several concepts and frameworks that guided our work.

2.1 Related Work: Decomposition

Standards [3], frameworks [13], and curricula [18] have included decomposition as an important idea, but much of the research on

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICER '18, August 13–15, 2018, Espoo, Finland

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-5628-2/18/08...\$15.00

<https://doi.org/10.1145/3230977.3230979>

assessing and understanding how students learn CS concepts in K-8 has focused on the use of particular programming concepts such as loops, conditionals, variables, and initialization [14, 17, 37].

A few papers have focused specifically on decomposition. Franklin et al. [15] observed differences in how students decompose the steps to navigate a 2-d grid based on age and math skills, whereas Meerbaum-Salant, Armoni, and Ben-Ari [27] found that patterns students used in Scratch projects (bottom-up development and fine-grained programming) were at odds with accepted decomposition practices. Werner, Denner, and Campe [41] analyzed Scratch games, creating a Game Computational Sophistication measure, which has a relationship to the way students decomposed their games into code. Despite a relative lack of research *focused* on computer science decomposition, our literature search found 31 papers that had embedded learning goals related to decomposition.

The goal of this study is to take that body of work, with both its explicit and implicit treatments of student learning of decomposition, and use the embedded information and a theoretical framework to create a learning trajectory for Decomposition.

2.2 Related Work: Learning Trajectories

Simon [38] described a learning trajectory as "a prediction as to the path by which learning might proceed" (p. 135) via students' engagement in a particular activity. Several computer science education researchers have recently begun developing learning trajectories and progressions for CT-related ideas. Much of this work has used learning analytics or other forms of analysis of students' programming artifacts to detect patterns in student thinking [9, 10, 36, 37, 41]. The resulting LTs tend to address broad tasks and understandings, such as programming [9] and computational thinking as a whole [37]. In prior work, we created trajectories covering more specific components of CT: Sequence, Repetition, and Conditionals [32].

In order to develop LTs for a specific components of CT, including this LT for Decomposition, we took inspiration from work in mathematics education. Since the introduction of the LT construct in 1995, mathematics education researchers have developed and studied LTs for a number of specific mathematics topics, including spatial thinking [6], linear measurement [5], and equipartitioning [11]. When developing our methods for LT construction, we drew, in particular, from the work of Confrey, Maloney, and Corley [11], who developed a set of LTs by first clustering mathematics standards into trajectory clusters, then mapping each cluster onto empirical evidence of student learning drawn from their own research and other literature. We elaborate on how we adapted their process in the Methods section. A more thorough description is in [32].

2.3 Conceptual and Theoretical Background

As will be described further in the Methods section, our process of creating an LT for Decomposition applied ideas from three theories and frameworks: constructivism, the Use, Modify, Create framework, and dimensions of disciplinary practices. We provide some background on each of these below.

2.3.1 Constructivism. Constructivism is a theory of knowledge that says all information is mediated through a recipient's prior knowledge and thus transformed as it is processed and integrated [1]. That is, learners do not simply receive knowledge, but rather

construct it. According to constructivism, learning happens when experiences occur that are different than what is expected. These experiences cause disequilibrium that leads to adaptation [38]. Hence, pedagogy based in a constructivist perspective must always strive to take prior knowledge into account and consider how to introduce disequilibrium that results in adaptations to students' current knowledge. We applied constructivist ideas in the creation of our LT by placing the ideas closest to everyday knowledge we expect most students to possess near the start of the trajectory then building toward less familiar ideas. This gradual progression toward less familiar ideas is intended to help teachers and curriculum designers to strategically design activities that introduce manageable levels of disequilibrium in students' understanding.

2.3.2 Use, Modify, Create. The Use, Modify, Create pedagogical approach was introduced by Lee et al. [22] as a way to scaffold and build students' ability to apply computing concepts to programming. Students begin by using programs that demonstrate the concept under investigation. Next they modify programs that use the concept in order to exercise their knowledge and explore more fully how the concept works. Finally, they create their own projects that incorporate the core ideas, requiring them to choose when the concept is appropriate and how to apply it. We applied the Use, Modify, Create pedagogical approach to the creation of our LT by placing goals related to using something created through decomposition first, followed by goals focused on the modification such a component, with goals focused on choosing when and how to create such components at the end.

2.3.3 Dimensions of Disciplinary Practices. We consider decomposition to be not only a content topic in computer science, but a disciplinary practice of computer science. As such, our work takes inspiration from work of researchers who have created learning progressions for practices in other disciplines. In Schwarz et al.'s [35] work on scientific modeling and Berland and McNeill's [8] work on scientific argumentation, the learning progressions are described not as a single sequence of levels of understanding, but as a set of dimensions of the practice. The dimensions of a practice are components that are related, but can develop separately. For example, Berland and McNeill's [8] dimensions of argumentation include the *argumentative product* and the *argumentative process*. Based on this prior work in progressions for scientific practices [8, 35], our methods for organizing existing knowledge about decomposition included attention to defining and connecting dimensions of the practice.

3 METHODS

We approached the creation of our Decomposition LT in three steps modeled after the work of Confrey, Maloney, and Corley [11]. First, we reviewed existing literature to identify learning goals. Next, we synthesized these goals to define a set of "stopping points" along our predicted learning path. We call these stopping points *consensus goals*. Third, we used information from the synthesized literature, as well as several conceptual frameworks, to make reasoned predictions about productive pathways students could follow through the consensus goals. Each of these steps is elaborated below.

3.1 Identifying Learning Goals

As a first step in trajectory building, Confrey, Maloney, and Corley [11] grouped a widely accepted set of grade-specific mathematics standards into "trajectory clusters" of related goals. In the absence of a widely accepted, grade-specific set of standards for CS/CT in grades K–8, we began our work by compiling a list of learning goals that had been discussed in extant computer science education literature. We searched the Educational Research Information Center database and the Special Interest Group in Computer Science Education (SIGCSE) and Innovation and Technology in Computer Science Education (ITiCSE) conference proceedings using keywords such as "computational thinking," "computer science domains," and "computer science pedagogy," with the additional signifiers "K–8," "K–5," and "K–12." We reviewed 108 scholarly articles that resulted from this search to identify learning goals (LGs) discussed by the researchers. This review produced 671 LGs. Further details about the review can be found in [31, 32]. Using keyword searches, we organized the LGs into clusters that related to particular CT topics. For the remainder of this paper, we focus on the cluster of LGs related to decomposition, which contained 63 LGs.

3.2 Synthesizing Consensus Goals

Our next step was to synthesize these 63 LGs into a manageable set of goals. The four authors of this paper, two with computer science expertise and two with curriculum development expertise, independently sorted the LGs into collections with common themes. Each of us articulated the themes as *consensus goals* (CGs), or statements intended to capture the consensus idea captured by the LGs in a collection. By comparing and discussing each others' sorts and CGs, we collaboratively came to agreement on a set of CGs that described the ideas captured in the cluster of decomposition LGs. This set of CGs served as the components of the LT to be connected in the next step.

3.3 Connecting Consensus Goals

With a set consensus goals defined, we next turned to defining pathways among the CGs. We defined these pathways by drawing arrows between pairs of CGs. Before describing our four heuristics for drawing arrows, we briefly describe our intended meaning for an arrow.

As noted above, Simon [38] describes a learning trajectory as a prediction about a possible learning pathway. As such, we conceptualize our arrows not as absolute dependencies among CGs, but rather as research- and theory-based predictions about how learning *could* proceed. Our aim is not to definitively prescribe the order or manner in which students learn content. Rather, we aim to apply existing knowledge from literature and existing conceptual frameworks to make reasonable suggestions that can guide curriculum design, teaching, differentiation, and assessment. However, we acknowledge that curricular order is a relationship between teachers, students, and the material. Students may take many different pathways, and even regress on our trajectory, as they develop their understanding [7, 19]. Likewise, teachers may have a different preference based on their understanding of the material.

With this overall aim in mind, we drew arrows between the consensus goals in four steps. First, we returned to the literature base

Table 1: Four LGs synthesized into one Decomposition CG

LG1	"[L]earn that the GUI is composed of different types of components". [29]
LG2	Break existing games into parts according to the types of computational thinking applied. [20]
LG3	"Identify the constituent elements of a system". [40]
LG4	Break "a story down into simple steps" at the appropriate "level of detail." [12]
CG	Systems are made up of smaller parts. (U)

to consider how evidence in the reviewed articles might suggest connections between CGs. This step was modeled after the work of Confrey, Maloney, and Corley [11] who, after organizing mathematics standards into trajectory clusters, mapped their clusters onto existing empirical research. However, because specific research on how student understanding of decomposition progresses is scant, this mapping could only serve as a starting point in articulating our pathways.

To further specify our trajectory, we next applied constructivist ideas to connect CGs into sequences that progressed from ideas we expect to be familiar to students toward less-familiar ideas. Then we applied Lee et al.'s [22] Use, Modify, Create framework to connect CGs. Lastly, we considered the dimensions of the practice of decomposition that were emerging from our work and drew arrows to connect and further define the dimensions. The Results section illustrates the application of each step.

4 RESULTS

We describe our results in five sections that illustrate how the trajectory grew in complexity across its development. First, we describe the 13 consensus goals we synthesized from the pool of learning goals. Next, we describe how we used the limited available evidence from the literature to draw the first arrows connecting the CGs. Finally, we describe how we used three different heuristics based in constructivism, the Use, Modify, Create framework, and practice dimensions, respectively, to further specify connections in the trajectory.

4.1 Consensus Goals

Sixty-three learning goals were used to create 13 consensus goals in the Decomposition trajectory. Detailed discussion of the literature support underlying every CG is beyond the scope of this article. Instead, we provide the full list of supporting LGs for two CGs, choosing examples that illustrate some of the interpretive aspects of our process. After these two examples, we list all 13 CGs in a single table along with one supporting LG for each.

Table 1 shows the four learning goals that were synthesized into the consensus goal, "Systems are made up of smaller parts." Quotation marks around text in the LGs indicate when the text was taken directly from the source articles. The U appearing after the text of the CG in the table indicates that this CG can be addressed with offline, or unplugged, activities.

Table 2: Six LGs synthesized into one Decomposition CG

LG1	Understand "the usefulness of modular code and how abstraction helps with modularity of code". [18]
LG2	"Remix and reuse resources" previously created. [2]
LG3	"Identify common patterns between older and newer problem-solving tasks". [2]
LG4	Use "built in functions". [33]
LG5	Use "built-in functions" when creating a program. [41]
LG6	Use "built-in methods". [30]
CG	Often existing code from other programs can be used to solve parts of a decomposed problem. (P)

The example in Table 1 illustrates the kinds of LGs used to support early goals in the trajectory. Within the CS-education literature, decomposition is typically discussed in terms of breaking apart *problems* or *tasks* into manageable parts [2, 4, 18, 26]. This idea is captured in a different CG in our Decomposition trajectory: "Complex *problems* can be broken into smaller parts." The CG shown in Table 1, "Systems are made up of smaller parts," represents a prerequisite to decomposition of a problem: understanding that many things in the world (not only problems) can be viewed as made up of parts. We chose "systems" as an umbrella term for these many things, from Weintrop et al. [40] (see LG3). As added support for the CG, we include LGs that suggest that students should identify parts within different examples of systems, including a GUI (LG1), a game (LG2), and a story (LG4).

Table 2 shows the six learning goals that were synthesized into the consensus goal, "Often existing code from other programs can be used to solve parts of a decomposed problem." The P appearing after the CG text in the table indicates that this is a programming-specific goal.

This example illustrates the way in which LGs were sometimes considered collectively, rather than individually, to form support for the CG. LG1 and LG2 directly support the CG because they directly address the utility of reusing code across programs. By contrast, LG3–LG6 support the CG collectively. On their own, they do not capture the idea in the CG. Identifying patterns, as noted in LG3, does not necessarily lead to reuse of code, and using a function or method, as in LG4–LG6, does not necessarily imply that the same code is being used for multiple purposes. However, when viewed collectively, LG3–LG6 capture the idea of matching a part of a current problem to a function used previously. This collective interpretation is close to the idea expressed in the CG.

All 13 CGs are shown in the first column in Table 3. The second column contains comments that help to differentiate the CGs from each other. The third column lists one learning goal from the literature we identified as support for the consensus goal. The final column lists the number of LGs that underlie the CG. Comparing the CGs to the example supporting LGs may help readers understand the manner in which we articulated big ideas from collections of specific student actions (the typical form of a learning goal in the literature). CGs that can be addressed through offline, or unplugged, activities are marked with a U in the table and shown in grey in the figures that follow. Programming-specific goals are marked with a P

and shown in white. Further information about the relationships among these CGs is included in the sections that follow, which detail how we connected the CGs into a trajectory.

4.2 Literature-Supported Connections

Figure 1 shows a subset of the decomposition consensus goals, connected with six arrows. In the text below, we describe how we used evidence from the literature to draw these six arrows.

Arrow 1a. In the K-6 CT framework posed by Angeli et al. [2], the act of breaking down a problem is identified as appropriate for students in K-2. By contrast, the act of assembling a solution from parts is identified as appropriate for students in grades 3-4. The authors of the framework therefore advocate a progression from top-down decomposition of a problem to bottom-up assembly of a solution. We used this information to draw an arrow from "Problem decomposition is a useful early step in problem solving." to "Often existing code from other programs can be used to solve parts of a decomposed problem." This arrow represents a transition from generalized, offline problem solving to specifically applying decomposition skills to programming.

Arrow 1b. In a five-day introductory course on parallel programming for fifth graders, Gregg, Tychonievich, Cohoon, and Hazelwood [16] used the pedagogical design of presenting students with sample projects to use as a basis or model for new projects: "Because this course was the first time most of the students had seen any programming language at all, we decided to provide a scaffolding in the form of example programs that they could look at and modify" (p. 54). At the end of the five day course, students enthusiastically presented projects that they had created either from scratch or by significantly modifying an example program. This result suggested the utility of asking students to modify code for particular purposes before using modified code as part of the solution to a broader problem. Thus, we used this information to draw an arrow from "Sometimes it is useful to modify and repurpose existing code." to the capstone goal, "Decomposition and modularization are useful in problem solving." Note that the capstone CG is designated as an unplugged goal to indicated that it is not exclusively about programming. After students have learned how to apply decomposition ideas in a programming context, they can learn to apply those skills to problems that are not exclusively computational in nature.

Arrows 1c and 1d. In her proposed chain of cognitive accomplishments students might go through during programming instruction, Linn [24] described the development of a collection of templates, or "stereotypic patterns of code using more than a single language feature" (p. 16), as an accomplishment that comes before the reuse of those templates. We see the recognition of patterns of code as stereotypic as akin to realizing that parts of code are non-unique. Relatedly, reusing those templates is similar to recognizing that existing code can be used to solve parts of other decomposed problems. Thus, we used this information to draw arrows from "System components are often not unique within or across systems." to "Code is reusable." to "Often existing code from other programs can be used to solve parts of a decomposed problem."

Arrow 1e. Several articles suggested that use of pre-existing functions should precede the creation and use of custom functions.

Table 3: The 13 Decomposition Consensus Goals

Consensus Goal	Comments	Example Supporting LG	No. of LGs
Systems are made up of smaller parts. (U)	See text supporting Table 1.	"Identify the constituent elements of a system". [40]	4
Complex problems can be broken into smaller parts. (U)	This CG identifies <i>problems</i> as a kind of system that can be decomposed.	Decompose a problem into "subproblems" that can be solved using computers. [40]	8
Problem decomposition is a useful early step in problem solving. (U)	This CG shifts from the simple fact of problem decomposition to its utility.	Recognize "the need for breaking down problems into smaller manageable tasks". [18]	6
Programs can be decomposed into components. (P)	This CG points out that <i>programs</i> , like problems, are systems that can be decomposed.	"[T]rack a program, capturing its various parts, but not the whole entity they form." [27]	5
System components are often not unique within or across systems. (U)	We identified non-uniqueness as a key idea underlying modularity.	Plan "with a sense of available templates in mind" to further reduce the task". [24]	1
Code is reusable. (P)	Reusability is another key idea underlying modularity.	Reuse modular parts of previously developed solutions. [40]	6
Often existing code from other programs can be used to solve parts of a decomposed problem. (P)	See text supporting Table 2.	Understand "the usefulness of modular code and how abstraction helps with modularity of code". [18]	6
Defining general procedures makes code more useful in the future. (P)	This CG calls out <i>creating</i> reusable parts as a separate skill from reusing.	"Develop solutions that consist of modular, reusable components". [40]	1
Sometimes it is useful to modify and repurpose existing code. (P)	This CG points out that code need not be reused exactly as given.	"[G]iven pieces of code" "modify for their own" program. [21]	2
A multi-instruction code snippet may serve a single, unified purpose. (P)	The idea of "chunking" code is a key idea underlying procedures.	Read a "function description presented in some language" to "understand what the function does". [28]	3
Code can be written in small parts (called procedures). (P)	This CG identifies procedures as a name for reusable code chunks.	"Code parts of a predesigned program to accomplish specific tasks." [17]	10
Defining procedures allows for easy reuse of code within programs. (P)	This CG connects procedures to reusability.	"[K]now how and when to create a separate method to accomplish a small, recurring task". [25]	10
Decomposition and modularization are useful in problem solving. (U)	This capstone CG ties all ideas in the trajectory together.	"Develop a solution by assembling together collections of smaller parts." [2]	9

Figure 1: Literature-supported connections among CGs

Figure 2: Constructivism-supported connections among CGs

One study found that middle school students spontaneously use built-in functions in Alice more than they created their own functions [33], another study described a pedagogical choice to place use of functions before creation of functions within a curriculum [30], and a third study placed use of functions (versus creation) in a lower level of an expert-developed framework used to analyze programming projects [41]. Using a pre-created function is an illustration that code is reusable, and so we used this evidence to draw an arrow from "Code is reusable." to "Defining procedures allows for easy reuse of code within programs."

Arrow 1f. Lee et al. [23] identified, as a common misconception, lack of understanding of the difference between a function definition (creation) and a function call (use). Students tended to omit the calls. We interpreted this to mean the act of creating a function is different from, and less difficult than, reusing that function within a program. We used this information to draw an arrow from "Code can be written in small parts (called procedures)." to "Defining procedures allows for easy reuse of code within programs."

4.3 Constructivism-Supported Connections

Figure 2 shows three more consensus goals and five additional arrows. These arrows were added based on the constructivist idea that learning should build on existing knowledge. Applying this idea to our particular context, we aimed to place the consensus goals most closely resembling everyday knowledge at the beginning of our trajectory, followed by CT ideas that may apply outside of programming, and then followed by programming-specific ideas.

The consensus goal most closely related to everyday knowledge is, "Systems are made up of smaller parts." As such, we placed this CG at the earliest place in the trajectory. This big idea is a natural stepping off point to three more computational-thinking specific ideas. Arrow 2a shows a progression from breaking *any system* into parts (recall from above that we use the word system to refer to any story, object, or other item that could be broken into parts) to the computational thinking practice of breaking a complex *problem* into

parts. Arrow 2b similarly shows a progression from breaking any system into parts to breaking a *program* into parts. Arrow 2c shows a progression from breaking a system into parts to examining the parts more carefully to note that they are often not unique.

Arrows 2d and 2e are an extension of the progression from breaking down systems to breaking down problems. Once students have begun to focus specifically on problems, a natural next step is to think about *why* breaking down a problem is a useful thing to do (Arrow 2d). At this point students have transitioned from everyday thinking into the computational thinking practice of decomposition. Arrow 2e connects the CG focused on the utility of decomposition to the overall capstone goal of the LT. When students reach this CG, they can coordinate unplugged decomposition ideas with programming-specific ideas about reusability and modularization.

4.4 Connections Based on Use, Modify, Create

Constructivism helped us to position the CGs that can be addressed outside of programming in relation to the initial programming goals. To help us position the programming-specific CGs, we turned to Lee et al.'s [22] Use, Modify, Create framework. Figure 3 shows one additional CG and six additional arrows we placed by applying Use, Modify, Create as a heuristic.

Arrow 3a is an "onramp" to the Use, Modify, Create progression, as it represents a transition from simply understanding that "Programs can be decomposed into components." to realizing that those components are tools that can be *used* in other programs (i.e., "Code is reusable."). Arrow 3b acknowledges that students do not necessarily need to modify those reusable components or create their own components in order to use a decomposition strategy to assemble a solution. It connects the CG that corresponds with the *use* part of the framework directly to the capstone CG ("Decomposition and modularization are useful in problem solving.").

Arrow 3c corresponds to a progression from *use* to *modify*. Arrows 3d–3f bring an aspect of *create* into the trajectory, as they

Figure 3: Connections among CGs supported by the Use, Modify, Create framework

connect the creation of procedures with the process of implementing a modularized solution. We did not connect the CG related to modification ("Sometimes it is useful to modify and repurpose existing code.") to the CGs about creating procedures because we did not see the skill of modifying code as directly relating to creating functions. We chose to capture the *modify-to-create* aspect of the framework by showing separate paths to the capstone CG. We would imagine a beginner or intermediate path to the capstone goal to include use of modification, whereas an advanced path to the capstone goal would include creation.

4.5 Connections Based on Dimensions

After thinking about specific elements about the trajectory as described in the above sections, our final step was to step back and think about the trajectory holistically. Specifically, we looked for dimensions [8, 35] of the practice of decomposition evident within the developing trajectory. Our thinking about dimensions led to the last CG and three more arrows being added to the LT, as shown in Figure 4.

Through discussion we identified two dimensions of decomposition: (1) top-down planning (decomposition of problems) and (2) bottom-up assembly (creation and use of modular code). While this distinction was somewhat useful, the two categories are not easily separable; they intersect and depend on each other. For example, the CG "Programs can be decomposed into components." reflects top-down thinking, yet it provides a foundation for bottom-up assembly. This connection is reflected in Arrow 4a in Figure 4. Students think about how programs can be decomposed into chunks, and that feeds into their exploration of how writing chunks of code is an aspect of creating a modular solution.

A skill related to use and development of code chunks is reflected in the CG, "A multi-instruction code snippet may serve a single, unified purpose." Students must realize that code chunks can have a particular purpose to think about using them within solution. Arrows 4b and 4c show how this idea connects into the LT.

5 DISCUSSION

Here we discuss various potential uses of the Decomposition LT.

5.1 Guiding Future Research

We believe our Decomposition trajectory will serve as a useful frame for guiding future research. Our efforts toward defining connections among consensus goals raised a number of interesting questions. For example, is it necessary, useful, or inconsequential to ask students to discern the purpose of code snippets before writing functions? Is the reusability of code something that needs to be directly addressed before assembly of solutions from parts, or might it make more sense to ask students to assemble solutions from code chunks assumed to be one-time-use? There are many such questions to be answered. We hope our trajectory will inspire many questions as well as studies pursuing answers.

One area of particular interest for us is the relationship between the CT ideas of decomposition and sequence. This Decomposition LT focuses on structural decomposition of problems, with little attention to temporal decomposition. While developing this LT and our LT for Sequence [32], we noted that the act of decomposing a problem into parts was difficult to separate from the act of parsing a task into steps – the latter being a key sequence idea. One way to make sense of the difference is to think of decomposed parts of problems as tasks, which are then decomposed into steps. That is, the difference could be considered one of grain size, with decomposition "parts" being larger and themselves decomposable into sequence "parts." However, our assumed context of K-8 education makes this distinction lose some meaning. When imagining the types of problems we might ask our youngest students to decompose, steps or commands often are the reasonable size of a "part."

Later in the trajectory, when students start to write functions, the aforementioned distinction does make sense. Students write functions using sequence ideas, and the functions form the parts of the larger solution. Thus, our current thinking is that the earliest parts of the Decomposition trajectory may overlap with sequence

Figure 4: The full Decomposition trajectory, with connections based on dimensions highlighted

ideas in significant ways, but the two topics diverge as problems and solutions become more complex. We think this is an area that would benefit from further research into student thinking.

5.2 Professional Development and Curriculum

In recent years, LTs are being used as crucial components in teacher professional development. Sztajn, Confrey, Wilson, and Edginton [39] posited that "these trajectories can support growth in mathematical knowledge, selection of instructional tasks, interactions with students in classroom contexts, and use of students' responses to further learning" (p. 148). K-8 teachers seeking to integrate CT into mathematics and other subjects may be keen to apply this type of student-centered practice. We hope this Decomposition trajectory would take its place alongside other LTs as elementary teachers plan their curriculum and instruction.

Though not specifically targeted to the teacher audience, the CGs are expressed in appropriate detail to allow teachers to understand how decomposition in CS may manifest in the various elementary grade levels. Also exciting is the potential extension of these decomposition ideas into other subject areas. As discussed in section 4.1, the early CGs use umbrella terms like "systems" for many things in the world. This subset of early (unplugged) goals can easily be applied to ideas in other subject areas like science and art. The potential identification of such points of synergy between CT and other subjects should be a boon to teachers and curriculum developers seeking to integrate CT into existing course materials.

5.3 Formative Assessment and Differentiation

Finally, we think that educators may find our LT useful for formative assessment and differentiation. Although we do not see the LT as outlining definitive paths [7], we do hope that the CGs and the connections between them might serve as a useful lens for teachers in thinking about what aspects of decomposition their students understand, what aspects they do not yet grasp, and productive ways to build on existing knowledge to advance students' understanding.

For example, through questioning and observation, a teacher may discover that her students understand how to build solutions from modular parts, but tend to build all the parts from scratch rather than looking at the potential utility of previously created code. In such a case, our LT might be used as a guide for thinking about how to connect code reusability to the development of modular solutions. In addition, a more formal decomposition measure could be developed by revisiting Werner et al.'s [41] Game Computational Sophistication measure and using it alongside our LT.

6 CONCLUSION

In this paper, we present a learning trajectory for the CT practice of decomposition. The trajectory is the product of an extensive literature review of computational thinking and computer science education research in grades K-8. The literature has been analyzed by a multidisciplinary team to extract learning goals. The learning goals were then synthesized into 13 consensus goals that reflect the overarching ideas expressed by the large collection of learning goals. We describe our approach for drawing connections between goals to indicate logical progressions between CGs. The pathways are rooted in explicit literature references, constructivism ideals, the use-modify-create paradigm, and dimensions of disciplinary practices. This collective approach explores the many possible ways that students can identify and relate to decomposition from previous experiences both in and outside of a computing environment. This trajectory can be used to drive further explorations in the understanding and practice of decomposition for the development of curricular materials, assessment, and professional development.

ACKNOWLEDGMENTS

This material is based on work supported by the National Science Foundation under Award 1542828 and Award 1742466. 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] Edith Ackermann. 2001. Piaget's constructivism, Papert's constructionism: What's the difference? *Future of learning group publication* 5, 3 (2001), 438.
- [2] 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. *Journal of Educational Technology & Society* 19, 3 (2016), 47.
- [3] Computer Science Teachers Association. 2017. CSTA K-12 Computer Science Standards. (2017). Retrieved March 18, 2018 from <https://www.csteachers.org/page/standards>
- [4] 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 (2011), 48–54.
- [5] Jeffrey E Barrett, Julie Sarama, Douglas H Clements, Craig Cullen, Jenni McCool, Chepina Witkowski-Rumsey, and David Klanderman. 2012. Evaluating and improving a learning trajectory for linear measurement in elementary grades 2 and 3: A longitudinal study. *Mathematical Thinking and Learning* 14, 1 (2012), 28–54.
- [6] Michael T Battista. 2007. The development of geometric and spatial thinking. *Second handbook of research on mathematics teaching and learning* 2 (2007), 843–908.
- [7] 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.
- [8] Leema K Berland and Katherine L McNeill. 2010. A learning progression for scientific argumentation: Understanding student work and designing supportive instructional contexts. *Science Education* 94, 5 (2010), 765–793.
- [9] Matthew Berland, Taylor Martin, Tom Benton, Carmen Petrick Smith, and Don Davis. 2013. Using learning analytics to understand the learning pathways of novice programmers. *Journal of the Learning Sciences* 22, 4 (2013), 564–599.
- [10] Paulo Blikstein, Marcelo Worsley, Chris Piech, Mehran Sahami, Steven Cooper, and Daphne Koller. 2014. Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences* 23, 4 (2014), 561–599.
- [11] 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.
- [12] Stephen Cooper, Susan H Rodger, Madeleine Schep, RoxAnn H Stalvey, and Wanda Dann. 2015. Growing a k-12 community of practice. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. ACM, 290–295.
- [13] K-12 Computer Science Framework. 2016. K-12 Computer Science Framework. (2016). Retrieved March 18, 2018 from <https://k12cs.org/>
- [14] Diana Franklin, Charlotte Hill, Hilary A Dwyer, Alexandria K Hansen, Ashley Iveland, and Danielle B Harlow. 2016. Initialization in scratch: seeking knowledge transfer. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. ACM, 217–222.
- [15] 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*. ACM, 231–236.
- [16] 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.
- [17] 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*. ACM, 267–272.
- [18] Shuchi Grover, Roy Pea, and Stephen Cooper. 2015. Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education* 25, 2 (2015), 199–237.
- [19] David Hammer and Tiffany-Rose Sikorski. 2015. Implications of complexity for research on learning progressions. *Science Education* 99, 3 (2015), 424–431.
- [20] Andri Ioannidou, Vicki Bennett, Alexander Repenning, Kyu Han Koh, and Ashok Basawapatna. 2011. Computational Thinking Patterns. *Paper presented at the Annual Meeting of the American Educational Research Association* (2011).
- [21] Yasmin B Kafai and Quinn Burke. 2013. The social turn in K-12 programming: moving from computational thinking to computational participation. In *Proceedings of the 44th ACM technical symposium on computer science education*. ACM, 603–608.
- [22] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. 2011. Computational thinking for youth in practice. *ACM Inroads* 2, 1 (2011), 32–37.
- [23] Michael J Lee, Faezeh Bahmani, Irwin Kwan, Jilian LaFerte, Polina Charters, Amber Horvath, Fanny Luor, Jill Cao, Catherine Law, Michael Beswetherick, et al. 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.
- [24] Marcia C Linn. 1985. The cognitive consequences of programming instruction in classrooms. *Educational Researcher* 14, 5 (1985), 14–29.
- [25] Jiangjiang Liu, Cheng-Hsien Lin, Ethan Philip Hasson, and Zebulun David Barnett. 2011. Introducing computer science to K-12 through a summer computing workshop for teachers. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 389–394.
- [26] Linda Mannila, Valentina Dagiene, Barbara Demo, Natasa Grgurina, Claudio Mirolo, Lennart Rolandsson, and Amber Settle. 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*. ACM, 1–29.
- [27] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. 2011. Habits of programming in scratch. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. ACM, 168–172.
- [28] Osvaldo L Oliveira, Maria C Nicoletti, and Luis M del Val Cura. 2014. Quantitative correlation between ability to compute and student performance in a primary school. In *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 505–510.
- [29] Arno Pasternak and Jan Vahrenhold. 2012. Design and evaluation of a braided teaching course in sixth grade computer science education. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, 45–50.
- [30] Eileen M Peluso and Gene Sprechini. 2012. The impact of Alice on the Attitudes of High School students Toward Computing. *Journal for Computing Teachers* 7 (2012), 2012.
- [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] Kathryn M Rich, Carla Strickland, T Andrew Binkowski, Cheryl Moran, and Diana Franklin. 2017. K-8 Learning Trajectories Derived from Research Literature: Sequence, Repetition, Conditionals. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. ACM, 182–190.
- [33] Susan H Rodger, Jenna Hayes, Gaetjens Lezin, Henry Qin, Deborah Nelson, Ruth Tucker, Mercedes Lopez, Stephen Cooper, Wanda Dann, and Don Slater. 2009. Engaging middle school teachers and students with Alice in a diverse set of subjects. In *ACM SIGCSE Bulletin*, Vol. 41. ACM, 271–275.
- [34] 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*. ACM, 616–621.
- [35] Christina V Schwarz, Brian J Reiser, Elizabeth A Davis, Lisa Kenyon, Andres Achér, David Fortus, Yael Schwartz, Barbara Hug, and Joe Krajcik. 2009. Developing a learning progression for scientific modeling: Making scientific modeling accessible and meaningful for learners. *Journal of Research in Science Teaching* 46, 6 (2009), 632–654.
- [36] Linda Seiter. 2015. Using SOLO to classify the programming responses of primary grade students. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. ACM, 540–545.
- [37] 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*. ACM, 59–66.
- [38] Martin A Simon. 1995. Reconstructing mathematics pedagogy from a constructivist perspective. *Journal for research in mathematics education* (1995), 114–145.
- [39] Paola Sztajn, Jere Confrey, P Holt Wilson, and Cynthia Edgington. 2012. Learning trajectory based instruction: Toward a theory of teaching. *Educational Researcher* 41, 5 (2012), 147–156.
- [40] David Weintrop, Elham Beheshti, Michael Horn, Kai Orton, Kemi Jona, Laura Trouille, and Uri Wilensky. 2016. Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology* 25, 1 (2016), 127–147.
- [41] 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.