A Hybrid Model for Teaching Recursion

Omar AlZoubi
Computer Science
Carnegie Mellon University in Qatar
Doha, Qatar
oalzoubi@cmu.edu

Davide Fossati
Computer Science
Carnegie Mellon University in Qatar
Doha, Qatar
dfossati@cmu.edu

Barbara Di Eugenio
Computer Science
University of Illinois at Chicago
Chicago, USA
bdieugen@uic.edu

Nick Green
Computer Science
University of Illinois at Chicago
Chicago, USA
ngreen21@uic.edu

Mehrdad Alizadeh
Computer Science
University of Illinois at Chicago
Chicago, USA
maliza2@uic.edu

Rachel Harsley
Computer Science
University of Illinois at Chicago
Chicago, USA
rharsl2@uic.edu

ABSTRACT

Novice programmers struggle to understand the concept of recursion, partly because of unfamiliarity with recursive activities, difficulty with visualizing program execution, and difficulty understanding its back flow of control. In this paper we discuss the conceptual and program visualization approaches to teaching recursion. We also introduce our approach to teaching recursion in the ChiQat-Tutor system that relies on ideas from both approaches. ChiQat-Tutor will help Computer Science students learn recursion, develop accurate mental models of recursion, and serve as an effective visualization tool with which hidden contexts of recursion can become evident.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education —*Computer science education*

General Terms

Design, Experimentation

Keywords

Recursion; Intelligent Tutoring Systems; Mental Models

1. INTRODUCTION

Recursion is a fundamental concept in computer science: it is a mathematical concept, a programming technique, a way of expressing an algorithm, and a problem-solving approach [16]. It is an essential and powerful computational problem solving technique that involves breaking down of a problem into smaller sub-problems of the same kind. Thus said, such decomposition is not easily understood by novice students learning recursion. Many students and novice programmers seem to struggle to learn it [3]. Some computer science educators argue that recursion is an inherently difficult concept to master [17], and it is one of the universally most difficult concepts to teach [6].

Turbak et al. [22] stated that a key reason why recursion is viewed as a difficult concept is because it is traditionally taught after students built up preconceptions about self-referential process based on their experience with looping. Additionally, Tessler et al. [21] indicated that students sometimes have difficulty recognizing different invocations of the same function, and they get confused by the bookkeeping required for each recursive call. In particular, they struggle with unfamiliarity with recursive activities, the visualization of the program execution, the back-flow of control after reaching the base case, comparison to loop structures, and the lack of everyday analogies for recursion.

It is believed that in order to understand the process of recursion and to be able to write recursive code one must be able to visualize the nature of a problem and how solutions to smaller similar problems are combined to solve the original problem [2]. In the next section we review some of the approaches to teaching recursion, discussing the advantages and limitations of each approach. We also introduce our approach to teaching recursion in ChiQat-Tutor, a system that helps students learn basic data structures and algorithms [10, 1].

2. MODELS FOR TEACHING RECURSION

Tessler et al. [21] cited a number of approaches to teaching recursion. These include conceptual models of recursion and control flow, and the use of visual aids. Conceptual models are defined by teachers and are used as tools for understanding or teaching of a system. Conceptual models used for teaching recursion include mathematical induction, abstract and conceptual discussions of recursion, process tracing, and structure templates of recursive code [24]. On the other hand, the visual aids approach relies on the use of algorithm animation, program visualizations using video games of animated characters and robots.

SIGITE'15, September 30–October 3, 2015, Chicago, IL, USA.
© 2015 ACM. ISBN 978-1-4503-3835-6/15/09 ...$15.00.
DOI: http://dx.doi.org/10.1145/2808006.2808030 .

2.1 Conceptual Models

Conceptual models for teaching recursion are tools that aim at helping students formulate accurate mental models of the concept of recursion [9, 24]. Conceptual models include mathematical induction, process tracing, stack simulation, and structure templates of recursive code [24]. It is believed that if a person has a mental model of a process, s/he will be able to make predictions about the behavior of that process, although sometimes inaccurately [13]. Moreover, possession of a model will allow a person to debug the model when s/he is faced with counterexamples.

Early research on teaching recursion embraced the use of the mathematical induction model [5]. In this model students are taught recursion through the theory of recurrence relations [23]. Similarly, Lewis [15] examined the role of four modes of algebraic substitution techniques on students learning to trace linear recursion. The author hypothesize that differences between these techniques may help students understand the recursion process as it executed in computers. However, a possible limitation with this approach is that beginning computer science students might not possess the necessary level of mathematical skills required to develop a clear understanding of recursion [5, 23].

Some researchers believe that more emphasis should be put on the declarative, abstract level of problem decomposition rather than the computational model and program execution visualization [5, 8]. A divide-and-conquer strategy is applied at the problem level, regardless of the machine implementation. Ginat and Shifroni [8] found that the use of such model significantly enhanced recursive programs formulation ability of students in comparison to establishing comprehension via understanding of the process of recursion execution. On the other hand, Wu et al. [24] found that concrete conceptual model, which provide an appropriate level of details of the process of recursion, is better than abstract conceptual models for teaching recursion.

Kahney [13] identified a number of mental models of recursion possessed by both novices and expert programmers in the context of SOLO programming. These include; 1) The Loop model: novices are hypothesized to possess this model. They view recursive procedures as a single object instead of a series of new instantiations. 2) The Odd model: students acquired the notion that the flow of control statement, rather than the results of pattern matching, acts as the stopping condition for recursion. 3) The Syntactic ”Magic” model: a student is able to match on syntactic elements and their positions, and make predictions about the behavior of the recursive programs on this basis, but has no clear understanding of how recursion works. 4) The Copies model: students view recursive instantiations of a recursive procedure as copies, as opposed to a single object, as in the Loop model. It is interesting point that novices have the Loop mental model of recursion. Some research suggested that it is important to teach recursion before loops and not vice-versa [22].

Wilcocks and Sanders [23] extended the work of Kahney [13] and evaluated a number of conceptual models to teaching recursion. These include the mathematical, tree, copies, analogies, and graphical recursive structure models. They found that the Copies Model is more viable than the other models. It is believed that this Copies Model is what expert programmers have of recursion [13]. However, Kahney [13] cautioned that having acquired the Copies mental model is not sufficient to determine what students really know about recursion, since students can make predictions about recursive procedures behavior without fully understanding recursion.

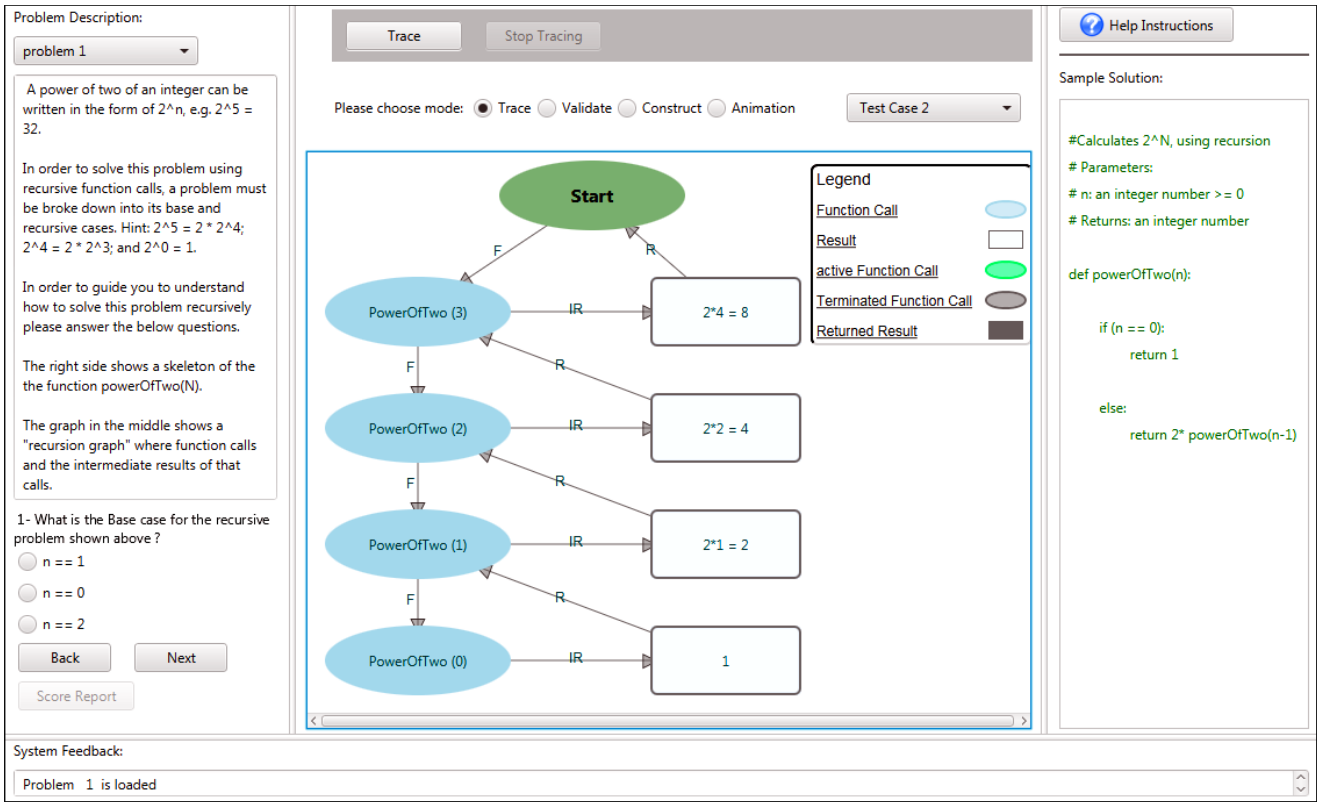
There has been some previous research on using structure templates and worked-out examples for teaching recursion in the context of LISP programming [17]. It was found that learning is facilitated by using abstract representations of the structure of recursion examples to guide initial coding attempts of students. The type of information exploited by this mechanism is an example or demonstration supplied by some source. However, Rinderknecht [18] mentioned that although examples can be used to develop analogical problem solving skills, care should be taken not to rely on them too early. This is in order to prevent students from developing the magic or syntactic model, identified by Kahney [13].

2.2 Program Visualization

Some researchers advocated program visualizations to introduce recursion. For example, Dann et al. [3] used program visualization technique to introduce recursion for students. They utilized a software system named Alice which is a 3-D interactive graphics programming environment. It has an object oriented flavor and it offers full scripting environment for 3-D object behavior (e.g., animals and vehicles) in a virtual world. Students can control object appearance and behavior by writing simple scripts. This allows students to gain intuitive sense and mathematical insight into the recursive process. However, Edgington [4] cites a number of limitations with the Alice system which included; a) Alice does not allow to create new objects programmatically; and b) Alice does not allow the inspection and modification of functions from within an executing program.

Similarly, Tessler et al. [21] used the Cargo-Bot video game for introducing recursion to novice programmers. In this game players control virtual robots by creating programs using a simple visual language. The goal is to control a robotic arm so that it moves a set of crates to a specified goal configuration. One of the interesting features of Cargo-Bot is that it supports recursion but not looping. The authors aimed to measure if students who played Cargo-Bot were able to transfer their experiences in playing Cargo-Bot to solve recursive problems in Java. Results from a controlled experiment showed significant improvements in students understanding of recursion using this technique.

Algorithm animation tools are commonly used as an aid to teaching recursion. The instructor programs an animation for commonly used algorithms (e.g. factorial, quick sort). The student runs the prepared animation observing the behavior using different inputs. Bower [2] argues that students should be able to manipulate the animations, not just watch them, in order to learn. It appears that there is no consensus on the benefits of algorithm animation as a learning aid. Stasko et al. [20] found no significant result suggesting that algorithm animators assist learning. They suggested that future research should focus on allowing students to construct their own animations. On the other hand, a meta-analysis of 24 experimental studies on the use of algorithm visualization (AV) as an aid for learning algorithms was conducted by Hundhausen et al. [12]. Their results showed that the effectiveness of AV is determined by how students use the AV technology rather than what the AV technology shows them. Other research showed that animation seems to make



A power of two of an integer can be written in the form of 2^n, e.g. 2^5 = 32.

In order to solve this problem using recursive function calls, a problem must be broke down into its base and recursive cases. Hint: 2^5 = 2 * 2^4; 2^4 = 2 * 2^3; and 2^0 = 1.

In order to guide you to understand how to solve this problem recursively please answer the below questions.

The right side shows a skeleton of the the function powerOfTwo(N).

The graph in the middle shows a "recursion graph" where function calls and the intermediate results of that calls.

1- What is the Base case for the recursive problem shown above ?

- n == 1
- n == 0
- n == 2

```
#Calculates 2^N, using recursion
# Parameters:
# n: an integer number >= 0
# Returns: an integer number

def powerOfTwo(n):

    if (n == 0):
        return 1

    else:
        return 2* powerOfTwo(n-1)
```

Problem 1 is loaded

Figure 1: Interface of ChiQat-Tutor. Left: a problem, its explanation, and questions. Center: task list and recursion graph. Right: help button and recursive code. Bottom: system's feedback.

a challenging algorithm more accessible, thus leading to enhanced learning [14].

Recursion Graphs (RGraphs) are visual aids used to introduce recursion to students [19]. RGraphs are similar to recursion trees as they show the invocation sequence of recursive functions. However they add the detailed calling sequences including intermediate results for each recursive call. An RGraph is a directed graph with two sets of vertices (oval for a recursive call, and square for pre/post processing statements of recursive calls). The RGraph is built layer by layer from top to bottom (i.e. breadth-first) with directed edges indicating the processing sequence. One particular feature of an RGraph is that it is traceable as it shows the detailed invocation sequence from one layer to another. Results showed that the use of RGraphs helped improve students' learning of recursion by providing flexibility in demonstration and more focused pedagogical interactions from students [19].

3. A HYBRID MODEL FOR TEACHING RECURSION IN CHIQAT-TUTOR

We believe that a hybrid approach, combining ideas from conceptual models and visual aids should be followed for teaching recursion, in order to capitalize on the advantages of both approaches, and avoiding misconceptions induced by either of them. In ChiQat-Tutor we developed an approach to teaching recursion that is based on the visual model of RGraphs, which is a clever visual representation of recursive execution. It also uses ideas from the conceptual model of teaching recursion in the form of code structure templates.

ChiQat-Tutor is a modular tutoring system whose goal is to facilitate learning of core CS data structures (e.g., linked lists, trees, stacks) and algorithmic strategies (e.g., recursion). The interface of ChiQat-Tutor is shown in Figure 1. We extended the use of RGraph in our system by implementing several interactive tasks described in Table 1. Students can interact with RGraph representations of different recursive problems (e.g., factorial, palindrome). The RGraph-based interactive tasks can help students identify the recursive structure of problems, understand recursive processes, and identify the critical features of recursive solutions to these problems. These tasks were designed with different levels of difficulty, which will allow users to progress smoothly from one task to another.

Previous research showed the importance of using graphical representations in the form of diagrammatic traces and animations of recursive problems, which allowed novices develop correct mental models of recursion [7, 23]. The tasks that can be carried on in ChiQat-Tutor combine characteristics from different conceptual models to teaching recursion. These include Stack Simulation, the Copies Model, and Tree Model [24, 25]. For example, the animation task represents a stack simulation approach to teaching recursion. Tracing, validating, and constructing RGrahps can enforce the Copies and Tree mental models of recursion. This allows students to look at recursive solutions from different angles. We believe that the use of RGraphs coupled with the different tasks that students can perform on them can help students develop the

Table 1: Task Description

Task Name	Description
Tracing	Users click on the nodes of the RGraph and follow the right order of execution. The nodes' color will change as users make progress.
Validating	Students work on two types of RGraph: an incomplete RGraph, and an RGraph that contains errors. Given a sample code, students are required to fill the partial RGraph, and then validate their solution. Similarly, they are required to correct the errors in the flawed RGraph, and then validate their solution.
Constructing	Learners build an RGraph for a given recursive code. The first few nodes of the RGraph are provided to them. Students need to validate their solution after finishing constructing the RGraph.
Animating	Learners play a prepared animation and observe the execution order of the recursive code. The node's color changes as the animation progresses; green indicates an active function call; gray indicates a terminated call or an intermediate result, which is explained in a legend next to the RGraph.
Answering Questions	Students answer multiple choice questions related to the current recursive problem. This task is designed to test learners' under- standing of the recursive problem and recursion in general.
Code Templates	Students are required to construct a recursive solution for a given problem. They are given a code template, and they can choose from a set of predefined code statements to construct their solution. Feedback can be given during the process.
Object Manipulation	Students can work on Linked Lists objects and Binary Search Trees. They can perform tasks such as traversing, sorting, addition and deletion of nodes. These tasks can be undertaken by constructing solutions based on a code template. The effect of their solution can then be seen on the object they working on, either a linked list, or a binary search tree.

Copies Model that expert programmers are hypothesized to possess.

Our approach for teaching recursion also embodies a much widely used strategy of using structure templates of recursive code and worked-out examples as an instructional material [25, 17]. The cognitive processes required to learn from a solved example are well understood. The learner must: (a) store in procedural memory the steps in the example; (b) interpolate the missing steps, since solved examples are necessarily incomplete to some extent; (c) infer the purpose of at least the main steps in the example; and (d) generalize over the specifics of the example. Working through examples (with the implicit or explicit expectation the learner will generalize from these examples) is pervasive in instructional situations, and much appreciated by students.

It is also important to mention that many operations on Linked Lists and Binary Search Trees (BST) are naturally performed with recursive solutions. Thus, improving students understanding of recursion will allow them to apply recursive solutions to these types of data structures. We plan to add tasks that allow students to manipulate Linked Lists and BSTs using recursion. These tasks include traversing, addition and deletion of nodes. ChiQat-Tutor will serve as an effective visualization tool with which hidden effects of nested function calls would become evident when applied to a whole range of problems that can be solved recursively.

4. EVALUATION

We have conducted a pilot study to evaluate the effectiveness of our system on learning recursion. The study looked for positive learning gains from pre-test to post-test after working with ChiQat-Tutor. Participants were 16 undergraduate students from Carnegie Mellon University in Qatar. The students were enrolled in "15-110: Principles of Computing", a course that teaches fundamentals of programming in Python. Students participated in the study after receiving an hour long lecture about recursion in class.

Table 2: Pre/post tests and learning gain results

	Pre-test	Post-test	Learning gain
Mean	.54	.60	.09
Standard deviation	.17	.13	.29
Range	.30 – .85	.45 – .85	- .50 – .50

4.1 Experimental protocol

First, students were asked to complete a consent form. Then they were asked to complete a pre-test which included problems that involved recursive decomposition using RGraphs. Then they were asked to work with ChiQat-Tutor for 30 minutes. There were 3 recursion problems to work with, each with 5 tasks, as described in Table 1. Students were then asked to complete the 3 recursion problems. Students were then asked to complete a post-test (identical to the pre-test) after finishing working on the problems.

4.2 Learning outcomes

Each question of the pre and post tests was graded by two graders on a scale from 0 to 5 following written guidelines. There were two questions per test, which brings the total for each test to 10. Scores were then scaled between 0 and 1. Hake's normalized learning gain (g) was then computed according to the formula in equation 1. Normalized gain is defined as the ratio of the difference in total score to the maximum possible increase in score [11].

$$g = \frac{post - pre}{1 - pre} \tag{1}$$

Table 3: Frequency of tasks attempted per student and the associated learning gain

ID	Animation	Questions	Tracing	Validating	Constructing	Learning gain
1	0	9	1	5	2	0
2	1	9	0	2	2	.50
3	0	9	0	1	1	.14
4	0	9	4	2	3	.29
5	0	5	1	0	1	-.13
6	2	10	0	2	1	.17
7	2	9	0	1	1	-.10
8	1	12	0	2	1	-.50
9	2	12	1	2	2	.50
10	0	21	2	2	2	.50
11	0	21	1	1	1	0
12	0	15	0	5	1	0
13	3	9	0	4	3	-.11
14	1	14	0	4	1	-.38
15	1	10	0	1	2	.20
16	2	12	1	7	3	.29

Table 4: Pearson correlation coefficient per task with learning gain

Animation	Questions	Tracing	Validating	Constructing
-.02	.12	.39	-.03	.48*

*. Correlation is significant at the .05 level (one-tailed)

Table 2 shows the mean and standard deviation for pre/post tests and learning gain. A paired t-test was conducted on pre/post tests scores to measure for any significant differences between the the tests scores. There was significant difference between the scores of pretest $(M = .54, SD = .17)$ and posttest $(M = .6, SD = .13); t(15) = 2.13, p < .05$.

We were interested in how individual tasks described in 3 influenced learning gains. This may influence future design decisions for the system. In order to do that, we counted the frequency of attempts for each individual task per student, throughout their interaction with the system. First, we computed the correlation between individual tasks and learning gain. Pearson correlation is shown in Table 4. It can be seen that the constructing task was significantly correlated with learning gain: r(14) = .48, p < .05. The tracing task was only marginally insignificant: r(14) = .39, p = .067. The other tasks did not show any significant correlations.

The constructing task is considered by far the most difficult task. We designed the tasks in such a way so students can work on simple tasks first and then progress to the most difficult. We assume that the animation, tracing, and validating tasks alone do not help students learn the copies model we wanted them to learn. In fact some of these tasks such as validating may encourage some students to learn the syntactic model of learning recursion. Furthermore, algorithmic animation in this experiment alone did not show to be helping students learning recursion. However we believe that it might help make an algorithmic concept more accessible. Answering questions showed little impact in helping students learn. However, we believe that the content and way of presenting these questions may have affected their contribution to learning gain.

5. CONCLUSIONS

We discussed the importance and difficulty of teaching recursion to novice Computer Science students. We also discussed different approaches to teaching recursion, with a focus on conceptual and visual tools. We then introduced our approach to teaching recursion in ChiQat-Tutor. We are currently using the first version of the recursion module in an introductory programming course at Carnegie Mellon University in Qatar enrolling approximately 60 students. Future work will be driven by the results of this first trial. We anticipate adding structured templates for writing code, scripting capabilities, intelligent feedback, and student modeling. We also plan to conduct larger studies to allow comparison between different conditions that include; the visualization approach, the conceptual approach, and the hybrid approach.

6. ACKNOWLEDGMENTS

This publication was made possible by NPRP grant 5–939–1–155 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

References

[1] O. AlZoubi, D. Fossati, B. D. Eugenio, and N. Green. ChiQat-Tutor: An integrated environment for learning recursion. In *ITS-AIEDCS 2014, 12th International Conference on Intelligent Tutoring Systems (ITS), 2nd Workshop on AI-supported Education for Computer Science (AIEDCS)*, Honolulu, HI, June 2014. Short paper.

[2] R. W. Bower. An investigation of a manipulative simulation in the learning of recursive programming, 1998.

[3] W. Dann, S. Cooper, and R. Pausch. Using visualization to teach novices recursion. In *6th Conference on Innovation and Technology in Computer Science Education*, ITiCSE '01, pages 109–112, New York, NY, USA, 2001. ACM.

[4] J. Edgington. Teaching and viewing recursion as delegation. *J. Comput. Sci. Coll.*, 23(1):241–246, 2007.

[5] G. Ford. An implementation-independent approach to teaching recursion. *ACM SIGCSE Bulletin*, 16(1): 213–216, 1984.

[6] J. Gal-Ezer and D. Harel. What (else) should cs educators know? *Communications of the ACM*, 41(9): 77–84, 1998.

[7] C. E. George. Experiences with novices: The importance of graphical representations in supporting mental models. In *12 th Annual Workshop of the Psychology of Programming Interest Group*, pages 33–44, 2000.

[8] D. Ginat and E. Shifroni. Teaching recursion in a procedural environment&mdash;how much should we emphasize the computing model? *SIGCSE Bull.*, 31 (1):127–131, Mar. 1999.

[9] T. Götschi, I. Sanders, and V. Galpin. Mental models of recursion. *SIGCSE Bull.*, 35(1):346–350, Jan. 2003.

[10] N. Green, O. AlZoubi, M. Alizadeh, B. D. Eugenio, D. Fossati, and R. Harsley. A scalable intelligent tutoring system framework for computer science education. In *CSEDU 2015, 7th International Conference on Computer Supported Education*, Lisbon, Portugal, May 2015.

[11] R. R. Hake. Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American journal of Physics*, 66(1):64–74, 1998.

[12] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3): 259–290, 2002.

[13] H. Kahney. What do novice programmers know about recursion. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '83, pages 235–239, New York, NY, USA, 1983. ACM.

[14] C. Kehoe, J. Stasko, and A. Taylor. Rethinking the evaluation of algorithm animations as learning aids: an observational study. *International Journal of Human-Computer Studies*, 54(2):265–284, 2001.

[15] C. M. Lewis. Exploring variation in students' correct traces of linear recursion. In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, ICER '14, pages 67–74, New York, NY, USA, 2014. ACM.

[16] D. D. McCracken. Ruminations on computer science curricula. *Communications of the ACM*, 30(1):3–5, 1987.

[17] P. L. Pirolli and J. R. Anderson. The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology/Revue canadienne de psychologie*, 39(2): 240, 1985.

[18] C. Rinderknecht. A survey on teaching and learning recursive programming. *Informatics in Education*, 13 (1):87–119, 2014.

[19] L. Sa and W.-J. Hsin. Traceable recursion with graphical illustration for novice programmers. *InSight: A Journal of Scholarly Teaching*, 5:54–62, 2010.

[20] J. Stasko, A. Badre, and C. Lewis. Do algorithm animations assist learning?: An empirical study and analysis. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 61–66, New York, NY, USA, 1993. ACM.

[21] J. Tessler, B. Beth, and C. Lin. Using cargo-bot to provide contextualized learning of recursion. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, ICER '13, pages 161–168, New York, NY, USA, 2013. ACM.

[22] F. Turbak, C. Royden, J. Stephan, and J. Herbst. Teaching recursion before loops in cs1. *Journal of Computing in Small Colleges*, 14(4):86–101, 1999.

[23] D. Wilcocks and I. Sanders. Animating recursion as an aid to instruction. *Computers & Education*, 23(3): 221 – 226, 1994.

[24] C.-C. Wu, N. B. Dale, and L. J. Bethel. Conceptual models and cognitive learning styles in teaching recursion. *SIGCSE Bull.*, 30(1):292–296, Mar. 1998.

[25] C.-C. Wu, G. C. Lee, and J. M.-C. Lin. Visualizing programming in recursion and linked lists. In *Proceedings of the 3rd Australasian Conference on Computer Science Education*, ACSE '98, pages 180–186, New York, NY, USA, 1998. ACM.