ChiQat-Tutor: An Integrated Environment for Learning Recursion

Omar AlZoubi[1], Davide Fossati[1], Barbara Di Eugenio[2], and Nick Green[2]

[1] Computer Science, Carnegie Mellon University
{oalzoubi,dfossati}@cmu.edu
[2] Computer Science, University of Illinois at Chicago
{bdieugen,ngreen21}@uic.edu

Abstract. Novice Computer Science (CS) students struggle learning recursion for reasons such as unfamiliarity with recursive thinking and difficulty in visualizing program execution. Many tasks in CS require a thorough understanding of recursion. We introduce the recursion module of ChiQat-Tutor, an environment for learning CS algorithms and data structures. ChiQat-Tutor uses the pedagogical tool of Recursion Graphs to help students visualize, manipulate, and learn recursive processes.

Keywords: Recursion, Recursion graphs, Intelligent tutoring

1 Introduction

Recursion is a fundamental and broad concept in computer science: it is a theoretical construct, a programming technique, a way of expressing algorithms, and a problem-solving approach [1]. Many problems in CS data structures, artificial intelligence, and algorithm analysis require a thorough understanding of recursion. Unfortunately, many students and novice programmers struggle learning it [2]. Computer science educators argue that recursion is an inherently difficult concept to master, and it is one of the most difficult to teach [3]. Turbak et al. [4] state that recursion is difficult because it is traditionally taught after students built up preconceptions based on their experience with loops. Tessler et al. [5] note that students have difficulty recognizing different invocations of the same function, and they get confused by the bookkeeping required for each recursive call. They particularly struggle because of unfamiliarity with recursive activities; visualization of program execution; back-flow of execution after reaching the base case; comparison to loop structures; and lack of everyday analogies for recursion. To understand the process of recursion and write recursive code one should be able to visualize the nature of a problem and how solutions to smaller similar problems are combined to solve the original one [6]. Turbak et al. [4] describe recursion as an instance of the problem solving strategy "divide, conquer, and glue (DCG)". In this strategy a problem is divided into simpler sub-problems, then solved (conquered) by yielding sub-solutions, and finally gluing these sub-solutions together into one. Recursion is an instance of DCG in which sub-problems are of the same kind as the original problem. Therefore, the function to solve these sub-problems is the same being defined to solve the whole problem. Such notion of function calling itself is confusing for many students.

We now present the recursion module of our new ChiQat-Tutor intelligent tutoring system. ChiQat-Tutor offers an environment for learning core CS topics such as linked lists, trees, and recursion. The system builds on iList [7] which was shown to significantly help students learn linked lists. ChiQat-Tutor will substantially expand the iList curriculum and add novel pedagogical strategies in CS education, including worked-out examples [8].

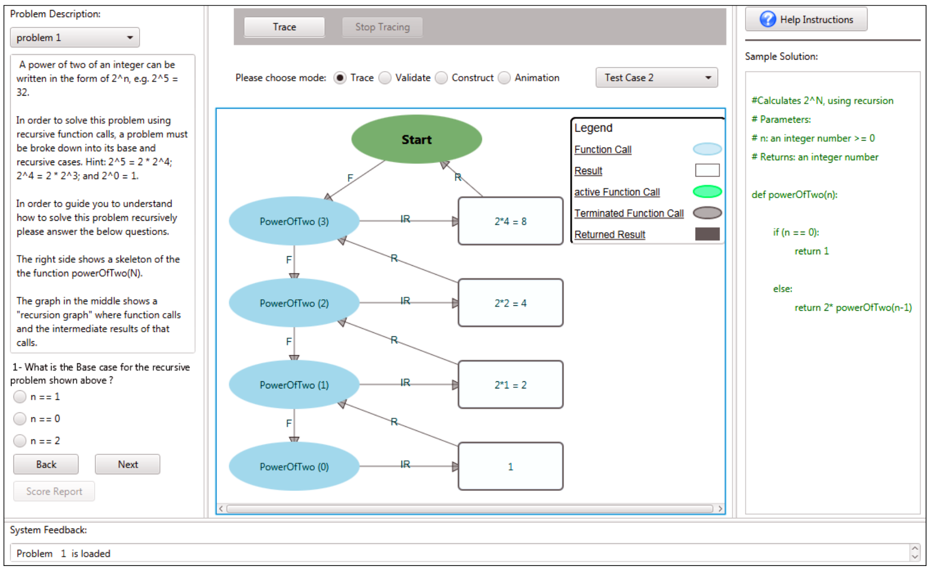
2 Approaches to Teaching Recursion

There are many approaches to teaching recursion. Tessler et al. [5] suggests conceptual models of recursion and control flow, and the use of visual aids. Conceptual models include mathematical induction, process tracing, stack simulation, and structure templates of recursive code [9]. Dann et al. [2] advocate program visualizations to introduce recursion, using their interactive programming environment Alice. Students control the appearance and behavior of 3D objects by writing simple scripts that allow them to gain insights into recursive procedures. For the same purpose, Tessler et al. [5] used the Cargo-Bot video game. In this game players control virtual robots by creating programs using a simple visual language. It is notable that Cargo-Bot supports recursion but not looping. According to the authors, results from a controlled experiment showed significant improvements in students' understanding of recursion.

Algorithm animation is yet another approach to teaching recursion. Instructors program animations of commonly used algorithms. Students run the animations and observe their behavior with different inputs. Bower et al. [6] argue that students should be able to manipulate the animations, not just watch them, in order to learn. In ChiQat-Tutor we make use of Recursion Graphs (RGraphs) which are a clever visual representation of recursive execution [10]. They are directed graphs with two sets of vertices (oval for a recursion call, and square for pre/post processing statements of recursive calls). An RGraph is built layer by layer from top to bottom with directed edges indicating the execution sequence. We extended the use of RGraph in our system by implementing several interactive tasks. Students will be able to interact with RGraph representation of different recursive problems to help them understand recursive processes.

3 ChiQat-Tutor: Recursion Module

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). In this section we focus on the new recursion module of ChiQat-Tutor. The recursion module provides learners with an interactive environment where they can perform a number of tasks using Recursion Graphs. Figure 1 shows its interface. Given a recursive problem (such as factorial or palindrome), the recursion module of ChiQat supports five individual tasks: (1) tracing an RGraph; (2) validating an RGraph; (3) constructing an RGraph; (4) animating an RGraph and (5) answering multiple choice reflective questions. These tasks differ in their difficulty, and learners are recommended to carry them in order.


Problem Description:

problem 1

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

Please choose mode: [✓ Trace] Validate Construct Animation

Test Case 2

Sample Solution:

```
#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)
```

System Feedback:

Problem 1 is loaded

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

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, then validate their solution. Similarly, they are required to correct the errors in the flawed RGraph, 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 nodes' color change 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' understanding of the recursive problem and recursion in general.

These tasks are inspired from the different teaching approaches to recursion we discussed earlier. Our system is based on the visual model of RGraphs. It also uses ideas from conceptual models of teaching recursion in the form of code templates and multiple choice reflective questions. The RGraph-based interactive tasks can help students identify the recursive structure of problems, and identify the critical features of recursive solutions to these problems.

4 Conclusions and Current Work

We discussed the importance and difficulty of teaching recursion to novice CS students. We also discussed different approaches to teaching recursion, with a focus on conceptual and visual tools. We then introduced ChiQat-Tutor, a novel environment for learning recursion. We are currently using the first version of the recursion module in an introductory programming course at Carnegie Mellon University enrolling approximately 60 students. Future work will be driven by the results of this first trial. We anticipate adding structured templates for writing code, intelligent feedback, and student modeling.

Acknowledgments. This work is supported by award NPRP 5–939–1–155 from the Qatar National Research Fund.

References

1. McCracken, D.D.: Ruminations on computer science curricula. Communications of the ACM **30**(1) (1987) 3–5
2. Dann, W., Cooper, S., Pausch, R.: Using visualization to teach novices recursion. In: 6th Conference on Innovation and Technology in Computer Science Education. ITiCSE '01, New York, NY, USA, ACM (2001) 109–112
3. Gal-Ezer, J., Harel, D.: What (else) should cs educators know? Communications of the ACM **41**(9) (1998) 77–84
4. Turbak, F., Royden, C., Stephan, J., Herbst, J.: Teaching recursion before loops in cs1. Journal of Computing in Small Colleges **14**(4) (1999) 86–101
5. Tessler, J., Beth, B., Lin, C.: Using cargo-bot to provide contextualized learning of recursion. In: Proceedings of the ninth annual international ACM conference on International computing education research, ACM (2013) 161–168
6. Bower, R.W.: An investigation of a manipulative simulation in the learning of recursive programming (1998)
7. Fossati, D., Di Eugenio, B., Ohlsson, S., Brown, C., Chen, L.: Data driven automatic feedback generation in the iList intelligent tutoring system. Technology, Instruction, Cognition, and Learning (TICL), Special Issue on Role of Data in Instructional Processes (2014) In press.
8. Di Eugenio, B., Chen, L., Green, N., Fossati, D., AlZoubi, O.: Worked out examples in computer science tutoring. In: AIED 2013, 16th International Conference on Artificial Intelligence in Education, Memphis, TN (July 2013) Short paper.
9. Wu, C.C., Dale, N.B., Bethel, L.J.: Conceptual models and cognitive learning styles in teaching recursion. SIGCSE Bull. **30**(1) (March 1998) 292–296
10. Hsin, W.J.: Teaching recursion using recursion graphs. Journal of Computing Sciences in Colleges **23**(4) (2008) 217–222