Reductive Thinking in Undergraduate CS Courses

Michal Armoni
Computer Science Department
The Open University of Israel
and School of Education,
Tel-Aviv University
972-9-778-1217
michal@openu.ac.il

Judith Gal-Ezer
Computer Science Department,
The Open University of Israel
108 Ravutski St.
Raanana, 43107, Israel
972-9-778-1353
galezer@cs.openu.ac.il

Orit Hazzan
Department of Education in
Technology and Science
Technion
Haifa
972-4-829-3107
oritha@tx.technion.ac.il

ABSTRACT

This paper describes research on the perception of undergraduate students of the concept of reduction. Specifically, based on an analysis of students' answers to questions addressing different CS topics, we present several findings regarding the ways in which undergraduate students conceive of and apply reduction. In addition to the research description and results, the paper discusses the role of reduction in CS and suggests several teaching applications.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer Science Education.*

General Terms

Theory.

Keywords

Reduction, Reductive Thinking, Computational Models, algorithms.

1. INTRODUCTION

Reduction is a problem-solving heuristic that characterizes both the theoretical aspects of computer science (CS), such as computability and algorithmics, as well as many other CS topics, such as software design. Essentially, solving a problem by reduction means transforming it into a simpler problem (or problems) for which a solution is already known, and constructing, or deducing, the solution to the original problem based on the solutions to the reduced-to problems.

Since a reductive solution uses known building stones, reductive strategy usually inspires less complicated solutions. For example, if in order to develop a solution for an algorithmic problem A, we can reduce problem A to another algorithmic problem B that has a known solution, then we can use B's solution as a black box, relying on the correctness of B's solution. In contrast, if we construct a new algorithm for A, we will have to prove the correctness of all its components (even when A's solution is based largely on B's known solution and only slightly alters it).

ITiCSE'06, June 26−28, 2006, Bologna, Italy.

Reductive thinking, as demonstrated by high school students' solutions to questions dealing with computational models, was discussed in [3]. The findings showed that many students preferred direct, non-reductive solutions, even in cases in which reductive solutions could have significantly decreased the complexity of the solution. It was also found that most of the students who constructed reductive solutions chose straightforward reductions, for which a lower level of reductive thinking is required, even if other, less straightforward, reductive solutions could have been more rewarding in terms of design complexity (the level of reductive thinking is defined as the conceptual gap between the original problem and the problem to which it is being reduced). These results motivated us to present a didactic strategy for the teaching of reductive thinking [2]. We demonstrated this strategy in [2] as applied to a course on computational models; it should probably be possible, however, to integrate it into the undergraduate CS curriculum in every context that lends itself to its use.

In [1] we discussed reductive thinking among undergraduate CS students in the context of a course on computational models. The findings presented indicated difficulties students encountered in applying reduction in the context of this course.

The research presented in the current paper continues the above-mentioned research works. Specifically, in this paper we present results of a research that addresses reductive thinking of undergraduate CS students in various contexts. Our goal in this research was twofold.

First, we examined the tendency of undergraduate CS students, at different stages of their studies, to use reductive solutions when solving problems taken from a variety of CS areas. The results of this research can teach us about the development of a reductive mode of thinking by undergraduate CS students. In addition, we checked whether students transfer their reductive thinking from the area of algorithmics, in which reduction is usually taught explicitly, to the field of computational models and the formal language theory, in which reduction is usually not taught explicitly despite the fact that it is an effective problem-solving heuristic in these areas. Studies in mathematics and in science education have shown that transfer – both inter-disciplinary and intra-disciplinary – is problematic (e.g., [11, 12]).

Second, since reductive thinking is a useful scientific problem-solving heuristic, and it is used widely in CS in particular, we

suggest that CS students learn this heuristic during their undergraduate studies. Accordingly, based on the results of this research, we intend to start evaluating didactic approaches aimed at developing reductive thinking. This stems from the fact that undergraduate CS students are usually exposed explicitly to the idea of reduction during their second year of study (in an algorithms course), or at a later stage (in a course on computability and complexity). However, as mentioned, reduction, as a problem-solving heuristic, is useful, important and effective for solving different kinds of problems (algorithmic problems as well as proofs) with different levels of complexity, starting from the CS1 course.

In Section 2 we elaborate on reductive thinking. In Section 3 we present the research setting. Section 4 presents and discusses the research results and based on these results, Section 5 suggests teaching applications. In Section 6 we conclude.

2. REDUCTION AS A HABIT OF MIND IN COMPUTER SCIENCE

Habits of mind are heuristics, problem-solving approaches [4]. In the context of CS, we talk about abstraction, reduction, successive refinements, etc., all of which are very helpful approaches in many problem-solving situations, but at the same time, cannot be conveyed by a rigorous set of rules that can be applied automatically in different concrete problem-solving situations. Rather, in order to apply them successfully, one must gain experience in their application, be aware of their potential contribution, and recognize which heuristic might be helpful in different situations.

Reduction is one of the most broadly useful habits of mind used to carry out arguments, mainly in computability theory. It seems to be one of those habits of mind that, we suggest, should be spanned over the entire undergraduate CS curriculum, as indeed is done in several courses. Among other advantages of this habit of mind, thinking in terms of reduction allows students to focus on structures in terms of their properties rather than their actual components. Not surprisingly, the centrality of reduction has been recognized in areas other than CS, for example in mathematical problem solving [13].

At the same time, however, it is all too common to find students who do well in a particular CS course but cannot apply their skills outside the narrow confines of the problems presented in that course. Specifically, while specific course material may be grasped and well-performed by students, if they fail to gain general habits of mind needed for solving problems in other topics of CS that require the same habits of mind, their performance may be less successful. Accordingly, our aim in educating our students, beyond teaching the actual CS material itself, is to impart the ability to employ various habits of mind that characterize one area of CS in more general contexts and, when required, to synthesize several of these ways of thinking.

Although reduction is a core CS concept, it is not an easy concept to teach. This fact can be explained mainly, but not only, by the fact that reduction is a "soft" concept – a concept that cannot be taught by rigorous formalism. In other words, unlike rigid concepts that can be characterized by rigid, formal rules, and similar to other soft concepts such as abstraction, recursion, encapsulation and programming paradigms, it is not sufficient to present a full, comprehensive and concrete definition of reduction, nor is it sufficient to lay out specific rules related to reduction. Furthermore, the difficulty in teaching soft concepts may be rooted in their *generality* – the fact that they can be applied in different domains with respect to different kinds of problems; at the same time, however, in order to be explained, they should also be illustrated by *specific* cases.

3. RESEARCH SETTING

Our research on reductive thinking among university students consists so far of two phases. The preliminary phase [1], focused on computational models with a population of 63 university students studying the course "Automata and Formal Languages". Findings were as follows: Students tend to use reductive solutions in the context of this course less than we had hoped they would. Furthermore, a substantial tendency toward direct, non-reductive solutions and solutions with a relatively low level of reductive characterization was identified.

The second phase of the research, which is described in this paper, focused on a variety of CS topics, starting from basic CS1 problems through algorithmic problems in different problem-solving situations, to computational model problems. This phase consisted of interviews with 19 students, in which students were asked to solve questions in the context of the above topics. The interviews enabled us to gain insights into the factors affecting students' choices of solutions and to establish connections between students' choices and the level of reductive thinking used in the solution process.

All students were interviewed towards the end of the academic year. Two groups of students were interviewed:

Group 1: Eleven of the students were freshmen. They were asked to solve four CS1 algorithmic problems.

Group 2: Eight students were at the end of their second year of study or in the middle of the third year and had all studied (or were about to finish studying) a course on algorithms as well as a course on computational models and the theory of formal languages. They were asked to solve five algorithmic problems at the algorithms course level, three computational model problems, and the same four CS1 problems given to the first group.

In addition, nine prospective CS high school teachers were asked to complete a questionnaire that presented the same four questions given to the first group of students.

For reasons of space limitations, we will not include here the interviews and questionnaire. They will be presented in a comprehensive paper on the topic. Interested readers are welcome to contact the authors and receive them by email.

4. RESULTS: STUDENTS' CONCEPTION OF REDUCTION

As mentioned before, reduction is a soft concept and therefore it cannot be presented by a set of rigid rules to be applied automatically in different problem-solving situations. In other words, is not always clear to students when and how to use reduction. In this spirit, the following research results are presented using a game metaphor, i.e., the game of applying reduction, addressing three main topics.

First, in Section 4.1, we address the very basic recognition of whether (or not) it is at all possible to play the game, i.e., we

describe situations in which reduction is recognized. Second, in Section 4.2, we discuss students' conceptions and beliefs regarding the legitimacy of playing the game, when conditions permit; specifically, how students conceive the actual use of reduction. Third, in Section 4.3 we address the actual play of the game, that is, the actual performance of reduction.

For reasons of space limitations:

- We will illustrate only some of the phenomena presented. Further illustrations will be presented in our talk.
- We will restrict the findings only to cases in which students either used (or even just considered using) reduction intuitively (first or second semester students) or were familiar with the concept. In the comprehensive paper mentioned above, currently in preparation, we present a full and comprehensive picture of students' conception of reduction, which also addresses other students' perceptions of the concept.

4.1. Recognizing reduction: Is reduction identified at all?

A necessary prerequisite for using reduction is the ability to identify relations and connections between different entities (problems, situations, conditions, etc.). This section addresses the tendency to recognize these events.

Our data analysis indicates that there is an evolving development of reductive thinking. Specifically, the research results indicate that first year students barely ever used reductive solutions, while the more mature students exhibited higher levels of awareness to the concept of reduction as well as to its potential use in different problem-solving situations.

Indeed, it is reasonable to assume that second semester students did not use reduction as a problem-solving heuristic simply because they had not yet been exposed to this approach explicitly, although indirect reference to reduction was made during their first year of study. Reduction is exhibited, for example, when concepts such as encapsulation and top-down design (demonstrated by procedures), and library units are taught, and at a later stage, through object-oriented principles.

Students' neglecting of reduction can be illustrated by their solution to the first of the four CS1 questions, in which they were asked to design an efficient algorithm that calculates the sum of all integer numbers between 1 and 100 that are indivisible by 3. Many students (not all of which were first year students) failed to recognize the connection between this problem and arithmetic series, even when it was clear that the resource of arithmetic series was available to them, since they had used it in solving other questions.

At the same time, some students recognized the option of using reduction, but in many cases did not exhaust its potential. With respect to the above-mentioned question, some did reduce this problem to the problem of calculating the difference between two series – that of all integer numbers between 1 and 100 and that of all numbers between 1 and 100 that are divisible by 3, recognized that the first series is an arithmetic series but failed to recognize that the second one is an arithmetic series as well.

As it turns out, the ability to identify such relations in which it is appropriate to use reduction, may depend on the problems and on the topics to which the problems refer. For example, all of the Group 2 students who used reduction tended to use it to solve problems that deal with shortest paths in graphs; but none of them used reduction to solve the following algorithmic problem:

> The input consists of a set of lists of various lengths. The lists should be merged into one list using a black-box merge algorithm that merges two lists and its cost is the sum of the lengths of the two merged lists. Design an efficient algorithm that determines the sequence of calls to the black-box merge algorithm such that the total cost of all merge operations is minimized.

Even though the algorithm designed for this problem by most students interviewed was essentially the same as the algorithm for constructing a Huffman tree for a given alphabet and its corresponding frequency list, none of the students identified a connection between this problem and Huffman code, not even at the reflection phase, after they finished designing the algorithm.

In another question, students were asked to design an efficient algorithm that constructs a minimal-weighted set of edges that contains at least one edge out of every circle in a given non-directed weighted graph. Most of the students who naturally used reduction for shortest-path problems could not identify the connection between this problem and spanning trees.

The effect of the topics to which the problems relate on students' ability to identify relations and on their tendency to use reduction in solving these problems leads to the issue of *transfer* [11, 12]. Reduction is mentioned explicitly in the teaching process of the algorithms course (usually in the tutorial sessions, in which reductive solutions were often demonstrated). At the same time, reduction is not usually mentioned in the teaching process of the course on automata and formal languages, although it is a powerful tool for solving problems related to formal language theory. The exposure to reduction in the algorithms course could have resulted in the development of a general tendency to reduction, a tendency that could be applied in other contexts as well. Our findings, however, indicate that there was no transfer of the tendency to use reduction, from the area of algorithmic problems to the area of computational models problems: All Group 2 students, even those who used reduction as a primary strategy for solving algorithmic problems, demonstrated a low level of reductive thinking when solving questions dealing with computational models.

4.2. Rules of the game: When is it legitimate to use reduction?

This section presents students' conceptions with respect to when reduction should be applied. In several cases and situations, for instance, students expressed a notion of illegitimacy with respect to the use of reduction. Since most of these students did indeed use reduction in some situations, this feeling of illegitimacy was found to depend on several factors:

A. Course framework: Some students considered reduction legitimate only if it was leaned in the same course; in other words, it bridges problems taught in the same course.

B. Question framework: Several first year students felt uncomfortable using reductions that span between different problems presented in different questions. They felt that the problems cannot be connected if they appear in different questions.

C. Black-box effect: Some students said that a black-box reduction solution is "cheating" or "stupid". This feeling also seemed to depend on some subjective factors. For some, it was not legitimate to use reduction to **simple** problems (such as maximum or sorting), whereas it was legitimate to use reduction to more complex problems, for which it seemed reasonable not to remember the detailed solution. Contrary to that, others felt that it was illegitimate to use reduction to previously-solved **complex** problems, if they did not remember exactly how these complex problems are solved. Such use of reduction was referred to as "cheating", since the main part of the solution was left hidden.

D. Setting: One student referred to the situation (interview, exam, homework, etc.) in which the question is solved: "It is OK to use reduction when solving home assignments, but in an interview, such as a job interview, it is illegitimate, since they probably want to know that I really know how to solve it".

The different factors that influence students' beliefs about the legitimacy of using reduction can be explained by the fact that they do not conceive of it as a rewarding problem-solving heuristic the use of which in fact reflects high problem-solving skills.

4.3. Playing reduction

We now present four phenomena that show how students apply reduction in different problem-solving situations. These phenomena suggest that students do not fully acknowledge the power of reduction.

A. Not exhausting the power of reduction when it is observed: Some students who used reduction for some problems did not exploit it in other cases, although they were already "half way there". For example, when asked to design an algorithm that finds the most frequent element in a given list of numbers, one of the students said that it is difficult since the list was not sorted, but he did not follow this flow of thoughts using reduction to sorting. In a few cases, when solving computational models problems, students identified a certain decomposition of a given language to sublanguages, but their solution eventually used either a less refined decomposition, or used no decomposition at all.

B. Preferring a lower-level rather than a higher-level reductive solution: In some cases, students considered a reductive solution but applied a direct solution or a solution leaning on a lower level of reductive thinking. Here are several examples:

1. Students sometimes felt that direct solutions were more efficient since they are tailored to the problem at hand, even when they knew that the two solutions – the direct and the reductive – have the same time complexity (and sometimes even the same actual time cost). For example, in one of the four CS1 questions, the students were asked to design an algorithm that finds the maximum element in a given list of numbers. In another question, they were asked to design an algorithm that finds the second maximum element in a given list of numbers. One of the students in Group 1 considered a reductive solution for the second question, in which the problem was reduced to the problem of finding the maximum element (in the original list, and then in the list obtained after deleting the maximum element). Then he claimed this solution to be non-efficient compared with the direct solution, in which two variables are used in one traversal of the list. He realized that the two solutions share the same time complexity, but was bothered by the existence of two loops in the reductive solution versus the one (more complex) loop in the direct solution.
2. For computational model problems, solutions based on constructive reduction were favored over solutions based on existential reduction (in which no automaton is constructed for the given language, but rather its existence is shown using closure properties). It seems that the constructive solution is perceived to be more complete, and was therefore favored, although the students were aware of the existential solution and of its validity.
3. In the course on algorithms, students learned that a maximum matching problem can be solved using reduction to a maximum flow problem. One of the home assignments included problems that could be solved using reduction to maximum matching. The teaching assistant reported that many students reduced the given problems to maximum flow problems, thus making it more concrete, by going deeper into the black box, and ending up with a more "complete" algorithm.

In several cases, students tended to underrate the difficulties they encountered while trying to solve a problem directly or using low-level reduction. After encountering significant difficulties along the solution process, and being confronted with the possibility of a reductive strategy which might have induced an easier solution, some students tend to state: "It was not as complicated", "I could have done it my way if I had worked on it a little longer", "I know how to do it", or "It could not made much of a difference". Such statements might reflect the conflict the students face between not acknowledging reduction as a rewarding problem-solving technique on the one hand, and the difficulties to handle details when reduction is not applied, on the other.

C. Reduction to a solution (rather than to a problem): This phenomenon addresses situations in which students explicitly say that they reduced the problem they were solving to a *specific solution* of another problem. One of the students said explicitly: "I am trying to think of relevant *algorithms* that I already know that can do something similar", rather than thinking of similar problems, as suggested by Polya [13].

Here are several examples:

1. Reducing the problem of finding the most frequent element in a given list of numbers to quicksort (or mergesort, or bubblesort in the case of other students), rather than to sorting.
2. Reducing the problem of finding a shortest s-t path in a {1, 2}-weighted undirected graph to BFS (rather than to the problem of finding a shortest path in a non-weighted graph).
3. Reducing the problem of finding the most frequent element in a given list of numbers to AVL-tree. (In fact, what this student was looking for in the AVL-tree data structure was the ability to extract a minimal element in O(logn) time, thus obtaining a sorted list in O(nlogn) time, what could obviously be achieved by reduction to sorting if only he had allowed himself to break free of the implementation details).

D. Reduction only when the details are known: In some cases, students used reduction to a problem only after they verified that they fully remembered the algorithm that solves that problem. The reduction was then formulated correctly as a reduction to a

problem. However, they could not establish the connection between the problems before they made sure they knew how the reduced-to problem is solved.

The essence of a black-box reduction is ignoring the solution of the problem to which the given problem is reduced and relying only on the knowledge that such a solution exists and can be obtained if necessary. However, we often saw difficulties relating to the black-box concept. Specifically, those difficulties described in paragraphs B, C and D regarding the black-box concept, indicate a tendency to reduce the level of abstraction, a behavior reported on in various contexts of mathematics and computer science [7, 8, 9]. This behavior is probably induced by the difficulty to handle abstractions meaningfully. The mental process of reducing abstraction makes the solution more concrete and thus more mentally accessible to these students.

5. TEACHING APPLICATIONS

Based on the above research findings, and based on our belief that reduction is an important problem-solving heuristic, we now present several teaching applications that may promote reductive thinking.

A. Start reductive thinking as early as possible in CS1: This recommendation is not limited, of course, only to reduction, but also to other heuristics and soft ideas such as abstraction, as well as to concepts such as efficiency [5, 6].

B. Demonstrate reduction in different situations and in various contexts: Instructors can relate explicitly to the reductive nature of solutions presented and to the advantages of these solutions over direct or lower-level reductive solutions. In this context, the didactic strategy for a course on computational models described in [2] might be helpful for other courses as well. We plan to adapt this strategy to other CS courses and to verify the effectiveness of this strategy in terms of developing reductive thinking.

C. Control the use of reduction: As suggested by Schoenfeld [14], control mechanism is an important ability in problem-solving situations. With respect to reduction, we should educate our students to be aware of how they actually use reduction. For example, on what assumptions they base their reductive solution, do they properly validate the correctness of the reduction, etc.

D. Course objectives: In general, although each course in the CS curriculum has a significant role and place, we suggest that, when appropriate, in addition to the course material, habits of mind will be highlighted as well. This perspective complements very well the idea that CS is in fact a unique problem-solving field.

6. CONCLUSION

Since CS is about transfer in the sense of building abstractions, it seems reasonable to look for ways that might help students develop modes of thought that are essential for doing CS. As far as we know, these skills, which computer scientists take for granted, are hardly ever discussed explicitly in CS courses.

Reduction might be one of these habits of mind that lead us to think about a problem in terms of bigger chunks. Such a perspective offers the thinker a more global view of the problem scene, and enhances the possibility for strategic planning and an intuitive feel for the problem. The reason for the difference between the two ways of thinking is, as usual, our limited capacity to simultaneously access large amounts of information from our short-term memory [10]. When dealing with a problem at the lowest level of detail (such as connections between arcs and nodes), we have no free space left for the broader picture, hence we tend to "lose the forest for the trees". In contrast, when we think in larger chunks, we might be temporarily relinquishing some of the precision, but in exchange gain on the global and intuitive side.

Acknowledgments

We would like to thank Yechiel Kimchi, Yuval Ishai and Dudu Amzallag from the Department of Computer Science of the Technion for their cooperation in the different stages of this research.

We would like also to express our thanks to the Technion's Fund for the Promotion of Research for its support of this research.

7. REFERENCES

[1] Armoni. M. and Gal-Ezer, J. (2006). Reduction – an abstract thinking pattern: The case of the computational models course. *Proc. of the 37th SIGCSE Technical Symposium on Computer Science Education*, pp. 389-393.

[2] Armoni, M. and Gal-Ezer, J. (2005). Teaching reductive thinking. *Mathematics and Computer Education*, 39(2), pp. 131-142.

[3] Armoni, M., Gal-Ezer, J. and Tirosh, D. (2005). Solving problems reductively. *Journal of Educational Computing Research,* 32(2), pp. 113-129.

[4] Cuoco, A., Goldenberg, E. P. and Mark, J. (1997). Habits of mind: An organizing principle for mathematics curriculum, *Journal of Mathematical Behavior*, 15(4), 375-402.

[5] Ginat, D. (1996). Efficiency of algorithms for programming beginners, *Proc. of the 27th ACM Computer Science Education Symposium*. New York, ACM Press, pp. 256-260.

[6] Ginat, D. (2001). Early algorithm efficiency with design patterns, *Computer Science Education* 11(2), pp. 89-109.

[7] Hazzan, O. (1999). Reducing abstraction level when learning abstract algebra concepts, *Educational Studies in Mathematics* **40**(1), pp. 71-90.

[8] Hazzan, O. (2003). How students attempt to reduce abstraction in the learning of mathematics and in the learning of computer science, *Computer Science Education* **13**(2), pp. 95-122.

[9] Hazzan, O. (2003). Reducing abstraction when learning computability theory, *Journal of Computers in Mathematics and Science Teaching* (JCMST) 22(2), pp. 95-117.

[10] Miller, G. A. (1956). The magical number seven plus or minus two: Some limits on our capacity for processing information, *Psychological Review* **63**, pp. 81-97.

[11] Noss, R., and Hoyles, C. (1996). *Windows on mathematical meanings*. Dordrecht, The Netherlands: Kluwer.

[12] Nunes, T., Schliemann, A.D., & Carraher, D.W. (1993). *Street mathematics and school mathematics*. Cambridge, UK: Cambridge University Press.

[13] Polya G. (1957). *How to solve it*, 2nd Ed., Princeton University Press.

[14] Schoenfeld, A. H. (1985). *Mathematical problem solving*. Orlando, FL: Academic Press.