J. EDUCATIONAL COMPUTING RESEARCH, Vol. 32(2) 113-129, 2005

SOLVING PROBLEMS REDUCTIVELY

MICHAL ARMONI
The Open University of Israel and School of Education, Tel-Aviv University

JUDITH GAL-EZER
The Open University of Israel

DINA TIROSH
Tel-Aviv University

ABSTRACT

Solving problems by reduction is an important issue in mathematics and science education in general (both in high school and in college or university) and particularly in computer science education. Developing reductive thinking patterns is an important goal in any scientific discipline, yet reduction is not an easy subject to cope with. Still, the use of reduction usually is insufficiently reflected in high school mathematics and science programs. Even in academic computer science programs the concept of reduction is mentioned explicitly only in advanced academic courses such as computability and complexity theory. However, reduction can be applied in other courses as well, even on the high school level. Specifically, in the field of computational models, reduction is an important method for solving design and proof problems. This study focuses on high school students studying the unit "computational models"—a unique unit, which is part of the new Israeli computer science high school curriculum. We examined whether high school students tend to solve problems dealing with computational models reductively, and if they do, what is the nature of their reductive solutions. To the best of our knowledge, the tendency to reductive thinking in theoretical computer science has not been studied before. Our findings show that even though many students use reduction, many others prefer non-reductive solutions, even when reduction can significantly decrease the technical complexity of the solution. We discuss these findings and suggest possible ways to improve reductive thinking.

INTRODUCTION

Reduction is an important method for solving problems in mathematics and other scientific disciplines. Essentially, solving a problem by reduction means transforming it into simpler problems (or problems for which the solution is already known), and constructing or deducing the solution of the original problem from the solution of the new problem.

For example, consider the Gauss method for solving linear equations. This is actually a reductive method, in which the matrix defined by the given set of linear equations is transformed into a triangular matrix, for which the solution is relatively simple. However, when this method is taught, its reductive nature is not usually emphasized. Consider another example, in which a student is asked to calculate the sum of all numbers between 1 and 101, which are not divisible by 3. A direct solution to this problem is very tedious and involves a large number of calculations. The problem can also be solved reductively: The required sum is the difference between the sum of all numbers between 1 and 101, and the sum of all numbers between 3 and 99 which are divisible by 3. Both sums can be calculated as arithmetic series. Thus the original problem can be reduced to two separate problems of calculating an arithmetic series. Students who are not used to reductive thinking may have difficulty reaching this solution, which has relatively low technical complexity.

In computer science, reduction is usually explicitly mentioned in the context of advanced topics traditionally taught in academic programs such as computability theory and complexity theory (Davis, Sigal, & Weyuker, 1994; Garey & Johnson, 1979). However, reduction can also be used in other fields of computer science, such as algorithmic design and computational models. It can also be used in high school curricula thus directing students toward reductive thinking at a relatively early stage of their scientific education.

In this article we discuss reduction in the context of the Israeli high school computer science (CS) curriculum, and specifically in the context of the learning process of computational models. The current high school CS program is based on a relatively new curriculum (Gal-Ezer, Beeri, Harel, & Yehudai, 1995; Gal-Ezer & Harel, 1999). One of the elective theoretical units of the curriculum is a unit on computational models (CM), for which a textbook and a teachers' guide were written (Armoni, Kaufman, & Bargury, 1998). The unit was developed by a team chaired by the first author, in consultation with the second. This unit addresses, in a relatively deep manner, several topics that relate to the theoretical foundations of computer science (e.g., computational limits, non-deterministic computational models and closure properties). Thus, it is considered a unique unit, since these topics are not usually included in computer science high school curricula.

The CM textbook presents reductive solutions to various problems in computational models. The study presented in this article examines to what extent high school students studying this unit use reductive solutions when solving questions related to computational models, and the nature of the reductions they use. We also discuss ways to improve reductive thinking. The issue of tendency to reductive thinking in the context of computational models has not been studied before, for high school, college and university students. This study is a part of a wider research we conducted on students' learning process when studying the CM unit. The research examined two additional issues: 1) The students' perception of non-determinism (Armoni & Gal-Ezer, 2003); 2) The general achievements of high school students studying the unit (Armoni & Gal-Ezer, 2004).

The rest of this article is organized as follows: In the second section we give a brief description of the CM unit and how reduction can be used to solve questions dealing with computational models. The third section is the main one, describing our study and its findings. Our conclusions and suggestions for further research are given in the last section.

REDUCTION IN THE COMPUTATIONAL MODELS UNIT

The Computational Models (CM) unit is planned for 90 hours, and taught over one school year. It is an elective unit and it is intended for students who have a specific interest in computer science and choose to study computer science for another year, beyond the basic courses. This unit covers both technical and theoretical issues concerning computational models. It introduces finite automata (deterministic and non-deterministic), pushdown automata and Turing machines.

The unit demonstrates and drills automata design, but it also discusses the theoretical properties of each model: computational power (in relation to previously introduced models), computational limits and closure properties. The topics in the CM unit are presented in a way that is suitable for high school students (Armoni & Gal-Ezer, 2003, 2004).

Most of the topics introduced in the CM unit are not usually covered in high school CS curricula; some of the technical issues that relate to constructing automata are sometimes touched upon but without discussing any theoretical aspects. For example, the high school computer science curriculum of ACM (Association for Computing Machinery) includes very few references to some of these, and then only as optional topics (Merritt et al., 1994). However, most college and university CS curricula (Atchison et al., 1968; Denning et al., 1989; Denning et al., 2002; Tucker et al., 1991) recognize these theoretical issues as fundamental to computer science. Therefore, the Israeli CS curriculum committee recommended exposing high school students to the topics included in the CM unit, to enable them to become familiar with some of the theoretical aspects of computer science, and to experience the kind of thinking that characterizes these aspects.

In the context of the CM unit, reduction can be used to solve design and proof problems. In a design problem, an automaton accepting a given, usually complex, language should be designed; while in a proof problem, a given language should be proved to be regular or context free.

When solving a design problem, the student can give a direct solution, in which a direct automaton is designed for the given language. For complex languages the design process may be complicated, and the student is likely to make errors in the construction.

Reduction can be used in the following way: First, the given language can be decomposed into simpler sublanguages, using operations such as union, intersection, concatenation, etc. Then the student should design an automaton for accepting each of these sublanguages, and finally the student should use known construction algorithms in order to combine the automata designed for the sublanguages into an automaton accepting the original language. We call this solution method *constructive reduction*. In constructive reduction, the designed automata are usually simpler than an automaton for the original language, and since the process of combining these automata into one automaton is algorithm-driven it does not require independent thinking.

When solving a proof problem, each of the above mentioned methods can be used: That is, a given language can be proved to be regular or context free by designing an automaton accepting it (a finite or a pushdown automaton, respectively), either directly, or by using constructive reduction. However, there is a third option: The given language can be decomposed into simpler sublanguages, as in constructive reduction, but then appropriate closure properties can be used to deduce that the original language is regular, or context free, without designing a specific automaton for it, but rather by proving its existence. We call this solution method *existential reduction*. Existential reduction is more abstract than constructive reduction, but it usually leads to shorter and less tedious solutions, since it enables to omit the combination process.

Both kinds of reduction are demonstrated in the CM textbook, and their advantages are discussed.

THE STUDY

This section consists of three subsections. The first describes the method and population of the study. The second and the third describe our findings regarding questions relating to regular languages and context free languages, respectively.

Method and Population

Developing the CM unit involved a three-year long experiment, during which the unit was taught in selected schools under the close supervision of the developing team. The majority of the population in this study are students who studied the CM unit in 1997-98, the third year of the experiment. The rest are students who studied the unit in 2000-01.

The entire research population included about 540 students, studying in 24 classes, in 10 schools, and taught by 15 teachers. About 160 of them were 11th graders, studying in 7 classes taught by 5 teachers, and 380 were 12th graders, studying 17 classes taught by 11 teachers. About 340 students studied mathematics on the highest level (5-point), about 170 studied mathematics on the 4-point level, and about 20 studied mathematics on the 3-point level (the lowest level).

Teachers were asked to include a number of questions provided by the developers on exams. Among these questions were one for each of the following chapters: Chapter 2, which introduces deterministic finite automata (DFA), Chapter 3, which discusses the properties of DFA, Chapter 4, which introduces additional models of finite automata—non-complete deterministic finite automata (NCDFA) and non-deterministic finite automata (NFA)—and discusses their properties, and Chapter 7, which introduces Turing machines. Another question related to Chapters 5 and 6, dealing with pushdown automata. The teachers were asked to send the students' full answers to these questions to the research team. At the end of the year they were asked to send the developers the students' answers on the final exam. Some of these questions were used to examine the tendency of the students to use reductive solutions and the nature of the reduction used in these solutions.

Reduction in Regular Languages

Eight questions were used to examine how the students use reduction for solving questions relating to regular languages: Two are questions which the teachers were asked to include in the exams given after the completion of the learning process of Chapters 3 and 4, respectively. Therefore, in solving the first of these two questions (Question 1), students could only use the DFA model, and while solving the second question (Question 2) they could also use the NCDFA and NFA models. We also used another question, given by one of the teachers after the completion of the learning process of Chapter 4, and five questions which were given in the final exams. Like in Question 2 above, in solving these six questions, students could use any model of finite automata: DFA, NCDFA or NFA. We shall first describe the students' answers to Question 1 (these answers were given after instruction that related only to the DFA model). Then we shall present the students' answers to Question 2 and 3 and to the other five questions (answers that were given after instruction that related to different types of FA models, including DFA, NCDFA and NFA).

The DFA Model

The below written question (Question 1) was included in the exam given after the completion of the learning process of Chapter 3:

Question 1

Let L be the language over the alphabet $\{a, b, c\}$ which contains all the words, exactly, for which at least one of the following conditions holds:

1. The number of *a*'s is equal to the number of *b*'s, and the sum of *a*'s and *b*'s is bounded by 6.
2. The word includes the pattern *abc* and ends with the pattern *bb*. Is *L* regular? Prove your claim.

This is a proof question, and can thus be solved either directly, or by using constructive or existential reduction. It has many possible solutions. A similar question was discussed in Armoni and Gal-Ezer (2005), as an example of a complex question relating to finite automata. However, the question presented there was a bit simpler, since the language presented there referred only to the first condition, and the solution presented there was not limited to the deterministic model, as is the case in this context. Here we will give only a short and general description, and refer the reader to Armoni and Gal-Ezer (2005), for detailed solutions. A direct automaton for this language is very large and complicated (since it should be deterministic) and contains at least 61 states. The students could not use regular expressions or the closure properties of regular languages under homomorphism and under inverse homomorphism, since these topics were not included in our high school computer science curriculum. Other reductive solutions for this problem can range from a solution in which the original language is decomposed into two sublanguages, one for each of the two conditions given in the language definition, and combining them by one union operation, to a solution in which the original language is decomposed into five very simple sublanguages, combined by 150 union and concatenation operations. The finer the decomposition, the simpler the technical complexity of the design process for the automata accepting the sublanguages. In all possible decompositions, using constructive reduction involves Cartesian-product construction algorithms and thus leads to automata with many states and transitions. We see that for this question neither a direct construction, nor a constructive reduction are reasonable solution methods to be used in an exam, and therefore this question directs the student to an existential reduction. Since almost all the students did indeed use existential reduction, in our analysis of the students' solutions to this question we focus on the level of the decomposition in the reduction used by the students.

The number of students who took an exam which included this question, and whose answer was sent to the research team is 394. These students studied in 21 different classes, in 9 schools, and were taught by 13 teachers; 266 of these students were 12th graders, and 128 were 11th graders; 94.5% of these students solved the question using reduction. The others gave direct (and incorrect) solutions (3%), or incorrect "proofs" of the regularity of the language (2.5%) (or its irregularity, in some cases).

If we analyze the solutions according to the level of decomposition, we get the distribution presented in Figure 1.

Fifty-four and eight-tenths percent of the students decomposed *L* into two sublanguages, corresponding to the two conditions in the definition of *L*, and using

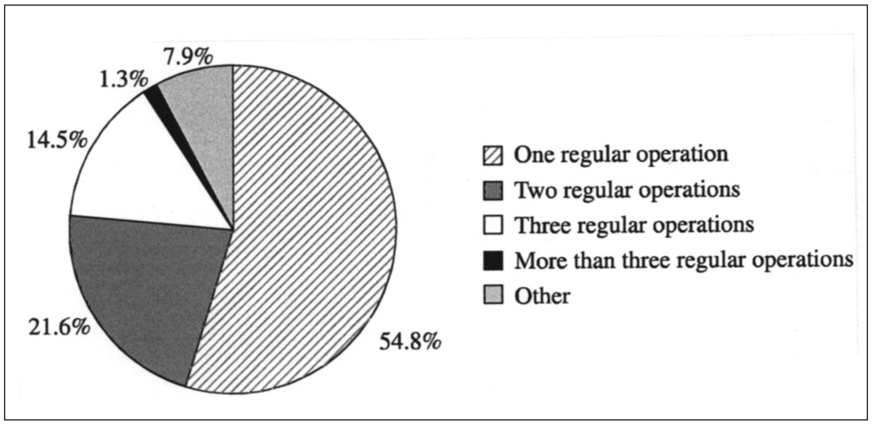

Figure 1. Solutions in the DFA model, analyzed by level of decomposition.

one union operation. Twenty-one and six-tenths percent decomposed the language corresponding to the second condition into two sublanguages, thus using three sublanguages, one union operation and one concatenation operation. Fourteen and a half percent of students decomposed the language corresponding to the first condition into two sublanguages (using one intersection operation): one that contains all the words in which the sum of *a*'s and *b*'s is bounded by 6, and another which contains all the words in which the number of *a*'s is equal to the number of *b*'s. Since the second one is an irregular language, these solutions were either incorrect or inaccurate (explaining informally why in spite of the irregularity of the second language, intersecting it with the first one yields a regular language).

Five students (1.3%) used more than three regular operations while decomposing the language. These decomposed the sublanguage corresponding to the first condition into four sublanguages: One that contains all the words which include no *a*'s and no *b*'s, one that contains all the words which include one *a* and one *b*, one that contains all the words which include two *a*'s and two *b*'s, and one that contains all the words which include three *a*'s and three *b*'s. Even though the decomposition in this solution is finer than those of the previous solutions, its technical complexity is essentially the same. The other solutions are either non-reductive solutions or non-complete reductive solutions for which the level of decomposition is not clear. None of the students solved the problem using the decomposition described in Armoni and Gal-Ezer (2005), in which the language corresponding to the first condition is decomposed into seven simple sublanguages, using three intersections and three unions, although the technical complexity of the solution induced by this decomposition is minor. The extreme solution described in Armoni and Gal-Ezer (2005) is not relevant in our case since it uses concatenation, an operation that was not taught in the third chapter of the CM unit.

To conclude, more than half of the students used the most simple and obvious decomposition, induced directly by the phrasing of the question, in which the language was defined using two numbered sub-conditions. About 36% of the students used a somewhat less simple decomposition, but one which is also almost directly induced by the phrasing of the question, using an intersection operation induced by the word "and" in sub-condition 1 or in sub-condition 2. Very few students went beyond the obvious decompositions and chose one which is not directly induced by the phrasing of the question, and none went far from this. No significant differences are found if the data is analyzed by mathematics level or grade.

Driven by our wish to develop reductive thinking, thinking which by nature should seek solutions in non-obvious places, we would like to see more students choosing decompositions which are not as obvious and directly induced by the phrasing of the question. The bright side of the coin is that a significant number of students chose a decomposition which was not the simplest possible.

If the data is analyzed by teachers, we can see that for some teachers, the distribution is significantly different. For example, one of the teachers emphasized reductive solutions as part of the teaching process, much more than other teachers. The distribution for this teacher is given in Figure 2, and it demonstrates better results, in the sense of reductive thinking, compared to the distribution of the entire population (Figure 1). Fewer students chose the obvious decomposition using one regular operation, and many more students chose decompositions using two and three regular operations. No students used more than three regular operations, or other solutions.

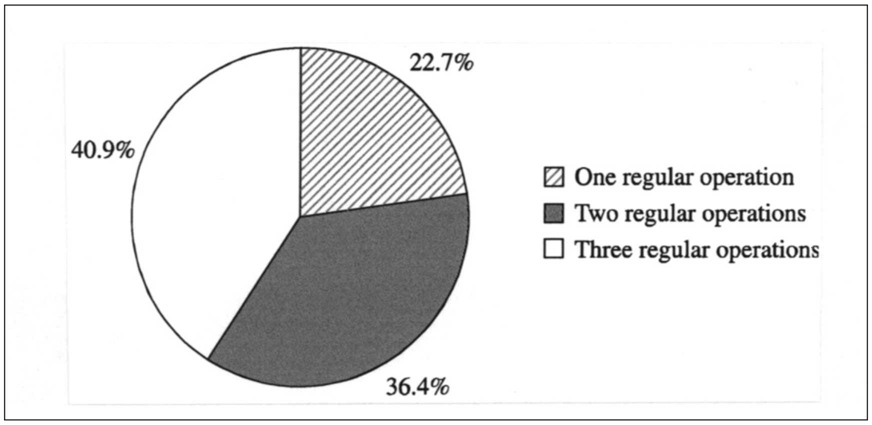

Figure 2. Solutions in the DFA model, Teacher 1.

The FA Model

Chapter 4 of the CM textbook introduces the NCDFA and the NFA models. After learning that these three models are equivalent, the student can use any of them to prove that a given language is regular, or to design an automaton for a given language.

In the context of reductive solutions, introducing the new models has a number of effects: On one hand, designing automata using the new models usually results in simpler automata, thus decreasing the advantage of reductive solutions. On the other hand, together with the new models, the student studied new regular operations (reverse and concatenation) with quite simple construction algorithms, and a simpler construction algorithm for the union operation. This enables more possibilities for reductive solutions, and decreases the technical complexity of reductive solutions which use constructive reduction with the new algorithms.

Seven questions were used to examine the tendency of students to use reduction when they are not limited to the deterministic model. Some were included in exams given immediately after the completion of the learning process of Chapter 4, and some were included in the final exams. We will focus here on two of these questions and describe our findings regarding the other questions briefly.

The teachers were asked to include the following question (Question 2) in the exam they gave on Chapter 4:

Question 2

Design an automaton that accepts the language over the alphabet $\{a, b, c\}$, that contains exactly the words for which at least one of the following conditions hold:

1. The word ends with the string *bc*.
2. The word consists of two parts: The first part contains the string *ba*, and the second part contains the string *ab*.

This is a design question, and can thus be solved using direct construction or by using constructive reduction. The number of students who took an exam which included this question and whose answer was sent to the research team is 339. These students studied in 17 classes, in 9 schools, taught by 11 teachers. Two hundred forty-four of these students were 12th graders, and 95 were 11th graders.

In Armoni and Gal-Ezer (2003), we analyzed the students' solutions to this question, but focused on the level of non-determinism. If we perform the analysis according to the reductive nature of the solutions we get the following results, presented in Figure 3.

Twenty-five and four-tenths percent of the students gave a direct solution; 41% used the obvious decomposition of two sublanguages, induced directly by the

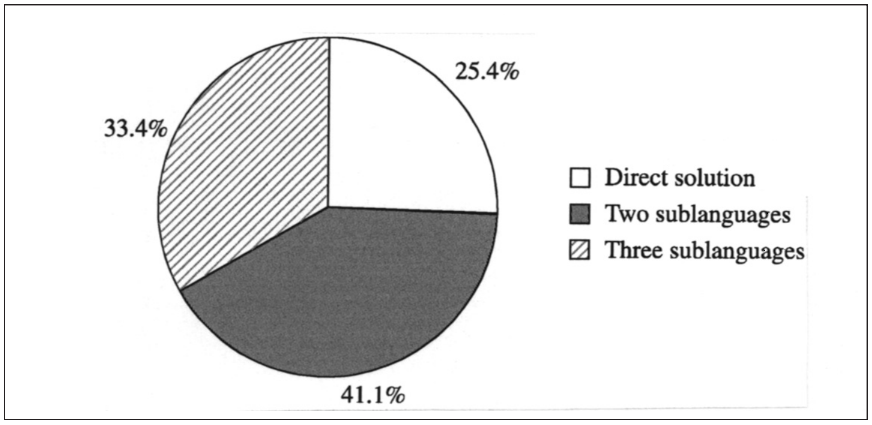

Figure 3. Solutions in the FA model.

phrasing of the question, in which the language was defined using two numbered sub-conditions; 33.3% decomposed the original language into three sublanguages, a decomposition induced by the word "and" in the second sub-condition. The fact that many students preferred a direct solution over a reductive one, even though the learning process of Chapter 4 reemphasized the advantages of reductive solutions, suggests that the tendency of students to use reduction depends on the complexity of the given language, which in this case is much simpler than *L* of the previous subsection. Again, the bright side of the coin is that a third of the students used the optimal decomposition for this question (which is much easier to find than the optimal decomposition for the previous question). For this question also, we seem to find evidence of teacher and teaching-process effect. Figure 4 presents the distribution for the students of the teacher mentioned above. Again, the results for these students seem much better than those of the whole population, and the fact that this teacher had strongly emphasized reductive solutions in the teaching process probably had a lot to do with it.

In the beginning of 2003, we conducted interviews with students regarding Question 2. The interviews strengthen the two assumptions mentioned above: that the tendency to use reduction depends on the complexity of the given language, and that the phrasing of the question affects the level of decomposition.

We interviewed four students who had finished studying Chapter 4 of the CM unit a few weeks before, and had been tested on the material a week before. Students on different levels of achievement were chosen by the teacher, who did not know in advance what question the students would be asked. The four students were asked to solve the question above. After completing their first version of the solution, they were asked a few questions regarding decisions they made when solving the problem.

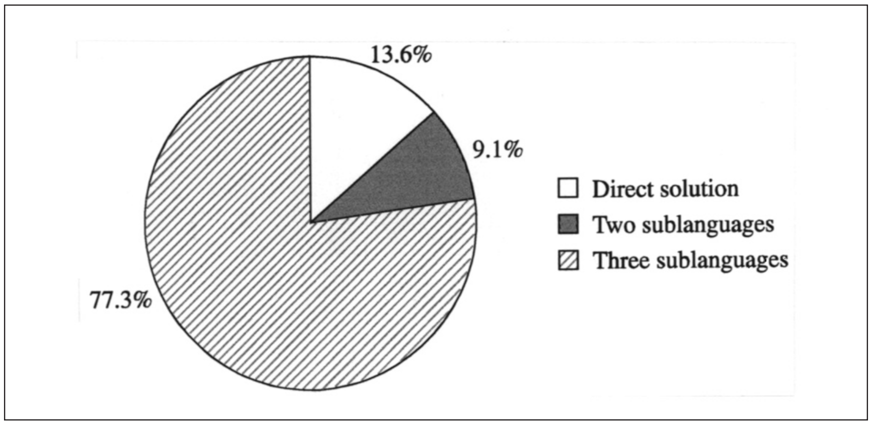

Figure 4. Solutions in the FA model, Teacher 1.

One of the students gave a reductive solution using two sublanguages, one gave a reductive solution using three sublanguages, one gave a direct solution, and another gave an allegedly direct solution, but which was induced by an implicit decomposition into two sublanguages (expressed in the design process of the automaton). All the students chose the solution method immediately after they finished reading the question, or even before. They devoted no time to considering the alternatives and weighing their advantages and disadvantages.

The second student was the only one who claimed to be driven by the desire to decompose the language as much as possible, though we cannot know what would be the result of this attitude when coping with a language for which the finer decompositions are not as explicit.

The first student said that he decomposed the language since this is the way they learned to solve such questions, in which a language is defined using a few conditions—defining a sublanguage for each explicit condition. Indeed, the decomposition used by this student corresponds directly to the definition of the language, which used two numbered sub-conditions. The other two students said that if the language had been more complicated they might have considered decomposition. These two students also referred to the phrasing of a given question as a factor which affects their choice of strategy:

- The phrase "design an automaton" (as opposed to a possible alternative phrasing such as "prove regularity of") motivates a direct solution, even though a constructive reduction can also be used to produce correct solutions corresponding such phrasing.
- If the definition of the language uses many explicit sub-conditions (three or more), it motivates a reductive solution.

The possible effect of the complexity of the given language on the tendency to use reduction is demonstrated again for the following question (Question 3), which was written by one of the teachers who participated in this research, and was included in an exam given after the completion of the learning process of Chapter 4.

Question 3

Let L_1 and L_2 be the following languages over the alphabet $\{a, b\}$: L_1 is the language of all the words which contain only an even number of b's. L_2 is $\{ba^n \mid n \geq 0\}$. What are L_1^2, $L_1 \cdot L_2$? Are they regular? Prove your claim.

This is an interesting question since, unlike in the previous problems, the decomposition and the operation it uses are already given, thus making a reductive solution much easier. One would expect that even though L_1^2 and $L_1 \cdot L_2$ are not very complicated, the students would use the decomposition handed to them, at least for $L_1 \cdot L_2$ (since $L_1^2 = L_1$). And yet, all nine students who solved this question ignored this information and none of them used a reductive solution. All of them designed direct automata for L_1^2 and $L_1 \cdot L_2$. Again, this suggests that the complexity of the given languages is a major factor in choosing the solution method. When the students feel they can cope with a direct solution, they prefer it to a reductive one.

The final exams also included questions relating to finite automata. We will give only a general description of our findings regarding the questions from the final exams: Four questions were proof problems, in which a complex language, usually defined by a number of conditions, was given, and the student was asked to determine if it is a regular language and to prove the claim. About 20% of 79 solutions were direct, (even though the given languages were quite complex), and almost all the reductive solutions used existential reductions. The fifth question was a proof problem, dealing with a simply-defined language, with no explicit sub-conditions. Almost 70% of the 136 students who solved this question used a direct solution, even though the technical complexity of a reductive solution for this question is significantly smaller. Again, it seems that the major factor affecting the tendency to use reduction is the complexity of the language definition: If the language's definition is long, and uses a few explicit sub-conditions, students tend more to solve it reductively (and usually define a sublanguage for each explicit sub-condition), while if the language's definition is relatively short and simple, students prefer direct solutions, although reductive solutions can significantly decrease the technical complexity even for simply-defined languages.

Reduction in Context Free Languages

Proof and design questions relating to context free languages usually also have a few possible solutions, induced by the kind of reduction used and the level of decomposition. Designing PDAs is usually more complicated than designing FAs, since the transition function ranges over triplets and not couples, and therefore decreasing the technical complexity using reduction may be significant.

After completing the teaching process of Chapters 5 and 6, dealing with PDAs and context free languages, the teachers were asked to include the following question (Question 4), consisting of two sub-problems, on the exam they gave on these chapters:

Question 4

1. Design a PDA accepting the language $\{a^n b^m c^m a^n \mid n,m \geq 0\}$ over the alphabet $\{a, b, c\}$.
2. Is the language $\{a^n b^m c^m a^k \mid n,m \geq 0, k > n\}$ over the alphabet $\{a, b, c\}$ context free? Prove your claim.

The first sub-problem is a design problem. The language's definition is relatively short and simple, but in a direct solution, the corresponding PDA is non-trivial and must be non-deterministic. There are also two possible reductive solutions, using one and two union operations, respectively. The decompositions used in these solutions are not directly induced by the phrasing of the question. They define sublanguages which handle special cases, such as $m = 0$ or $n = 0$, and a sublanguage for the non-special cases. The technical complexity of these solutions is much less than that of the direct one. However, all 309 students who solved this question and whose answers were sent to the research team, gave direct solutions to this sub-problem.

The second sub-problem is a proof problem, and again, the language is defined in a relatively short and simple manner. This sub-problem has a few possible solutions:

- A direct solution. The corresponding PDA is not based on the same algorithmic idea as the one corresponding to the first sub-problem. Thus, a student using the direct strategy starts almost from scratch, and does not use the automaton designed for the first sub-problem.
- A reductive solution which uses existential reduction. The original language is decomposed into two sublanguages, using one concatenation. The first sublanguage is the one from sub-problem 1, and the second is a simple regular language. The additional technical effort involved in this solution to sub-problem 2 is almost nil.
- A reductive solution which uses algorithm-driven constructive reduction. The decomposition is the same as that described in the previous solution. However, since this solution uses a construction algorithm which accepts as input two PDAs, a PDA must be designed for the second, regular, sublanguage (while in the previous solution it was enough to design an FA, thus proving that the sublanguage is regular and to deduce that it is context free due to the fact that regular languages are a subset of context free languages). This

solution was not really an option for the students studying the CM unit, since while the closure property of context free languages for concatenation is mentioned in the CM textbook, there is no complete proof, only an informal explanation describing the corresponding construction algorithm without fully presenting it.

- There is another reductive solution, which uses a free constructive reduction, rather than an algorithm-driven constructive reduction. The students can use the automaton designed for sub-problem 1, and transform it into an automaton for sub-problem 2, by making a few local changes. Since for context free languages only the closure property for union is fully proved in the CM textbook, while the other two properties discussed are either explained informally (for concatenation) or only mentioned with no explanation (for reverse, since the proof uses context free grammars which are not part of the CM syllabus), a free constructive reduction is sometimes the only constructive reduction possible.

The number of students who took an exam which included this question, and whose answers were sent to the research team is 309. These students studied in 17 classes, in 7 schools, taught by 9 teachers; 205 of these students were 12th graders, and 104 were 11th graders. Almost a quarter of them gave a direct solution. Almost half used constructive reduction, and only a third used existential reduction, for almost a quarter of the students with incorrect decomposition (this comes to more than 100% since some students gave more than one solution).

Similar, and even more obvious, are the findings for the questions relating to context-free languages in the final exams. These included 6 proof problems, all had reductive solutions with technical complexity significantly lower than that of the direct solutions, and for some there were a few relevant decompositions, for which the principle mentioned above usually held: The finer the decomposition, the smaller the technical complexity. About 40% of the students gave direct solutions; many of the students that used reduction, used constructive reductions even when existential reductions were sufficient; and most of the students did not use fine decompositions for questions which had more than one possible decomposition.

SUMMARY, CONCLUSIONS, AND FURTHER RESEARCH

Using reduction when solving design and proof problems relating to computational models simplifies the technical complexity of the solutions. Reductions which use finer decompositions induce even more technically simple solutions, especially if existential reduction is used instead of constructive reduction for solving proof problems.

Reductive solutions are demonstrated in the CM textbook, and their advantages are discussed. Indeed, we found that a substantial number of students used reductive solutions. However, many other students tended to give direct solutions, especially if the phrasing of the question led them to think that the given language is not complicated, and can be reasonably coped with using a direct strategy.

When using reduction, many students tended to use obvious decompositions, usually induced directly by the phrasing of the question, and not significantly decreasing the technical complexity. Our findings also showed that many students used constructive reduction even if existential reduction would have been sufficient, especially for questions relating to context free languages.

These tendencies may result from the relatively abstract nature of reductive solutions, as compared to direct solutions. That is, in a way, reductive thinking patterns require abandoning the regular, linear path that usually leads to a solution and viewing the problem in a different way, through a prism. Existential reduction is also more abstract in nature, as compared to constructive reduction, since it involves proving the existence of a solution without actually pointing at it. The effect of the abstract nature of reduction on the tendency to use it should be further studied.

For some of the questions, we also noticed a difference in the data, when analyzed by classes or by teachers. In some of the classes, students used reduction more, while other classes tended more to direct solutions. It seems that in classes where the teacher demonstrated and emphasized reduction, the students used reduction more. Our study did not focus on the teachers, and therefore this impression should be investigated in future studies. If this impression is verified, it suggests that emphasizing reduction during the teaching process may encourage students to use it more. Reduction can be emphasized in class by presenting questions together with a variety of possible solutions—direct solutions and reductive solutions with various levels of decomposition—thus demonstrating the advantages of reduction. We suggested a number of such questions in Armoni and Gal-Ezer (2005). Clearly, the effect of such teaching methods on the tendency of students to use reduction should be further studied.

Another interesting issue is that of transfer in the context of reduction, both within computer science and to other scientific disciplines. Within the discipline, it is interesting to examine whether students who were exposed to reductions in one field, such as computational models, tend to use reduction more often than students who were not exposed to reductive thinking, when solving questions in other areas of computer science. For example, do students who used reduction when studying computational models cope better with computational or polynomial reductions, when they later take a course in computability and complexity theory? A positive answer to this question would lead to the conclusion that in order to develop reductive thinking it is important to introduce reduction everywhere it fits in the CS curriculum (high school or academic curriculum), such as in a course about computational models.

In a broader, inter-disciplinary context, we might ask whether a student who was exposed to reductions in one scientific field, such as computational models in computer science, would tend to use reductive thinking patterns when coping with various issues in other scientific disciplines.

Studies in mathematics and in science education have shown that transfer—both inter-disciplinary and intra-disciplinary—is problematic (e.g., Glynn, 1991; Noss & Hoyles, 1996; Nunes, Schliemann & Carraher, 1993; Stavy & Tirosh, 1993). Do these findings hold for reduction as well? If transfer is problematic in the context of reduction, then including references to reduction is essential in every context that lends itself to its use. However, if students who acquired the type of thinking that characterizes the use of reduction in one domain tend to use it in other domains, then developing an inter-disciplinary unit which deals with reduction as a general scientific concept providing examples from various domains seems appropriate. All in all, it seems that encouraging students to apply reduction in solving and proving problems in various scientific domains is an important, non-trivial task. Thus, the impact of including references to reduction in various components of the scientific curriculum of various disciplines that lend themselves to using reduction, and the effect of including an interdisciplinary unit dealing with reduction in the high school curriculum on the development of reductive activities, need to be further explored.

Our findings show that, for many students, reductive thinking does not come naturally. However, it seems that students can be directed toward it. Therefore, when teaching mathematics and other scientific disciplines, teachers also should try to propose reductive solutions and discuss their advantages. When the syllabus includes solutions which are reductive (such as the Gauss method for solving linear equations), their reductive nature should be emphasized. Again, this attitude is even more important if transfer is found not to be achievable for reduction. In this case, a teacher cannot rely on the exposure to reduction in one field to promote reductive thinking in another, and must relate to reduction in every relevant field.

REFERENCES

Armoni, M., Kaufman, Y., & Bargury, I. (1998). *Computational models, a textbook and a teacher's guide*. Tel-Aviv: The Open University of Israel (in Hebrew).

Armoni, M., & Gal-Ezer, J. (2003). Non-determinism in CS high-school curricula. *Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference (FIE03)*, F2C-18-23.

Armoni, M., & Gal-Ezer, J. (2004). On the achievements of high school students studying computational models. *Proceedings of the 9th Annual SIGCSE conference on Innovation and Technology in Computer Science Education (ITiCSE04)*, 12-16.

Armoni, M., & Gal-Ezer, J. (2005). Teaching reductive thinking. *Mathematics and Computer Education*, in press.

Atchison, W. F., Schweppe, E. J., Viavant, W., Young, D. M. Jr., Conte, S. D., Hamblen, J. W., Hull, T. E., Keenan, T. A., Kehl, W. B., McCluskey, E. J., Navarro, S. O., & Rheinboldt, W. C. (1968). Curriculum ’68, recommendations for academic programs in computer science, *Communication of the ACM*, *11*, 151-197.

Davis, M. D., Sigal, R., & Weyuker, E. J. (1994). *Computability, complexity and languages, fundamentals of theoretical computer science*. New York: Academic Press.

Denning, P. J. (Chair). (2002). *IEEE Computer Society/ACM Task Force, Year 2001 model curricula for computing (CC2001)*. Available on line at: http://www.computer.org/education/cc2001/report/

Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., & Young, P. R. (1989) Computing as a Discipline. *Communication of the ACM*, *32*, 9-23.

Gal-Ezer, J., Beeri, C., Harel, D., & Yehudai, A. (1995). A high school program in computer science. *Computer*, *28*, 73-80.

Gal-Ezer, J., & Harel, D. (1999). Curriculum and course syllabi for a high-school program in computer science. *Computer Science Education*, *9*, 114-147.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to NP-completeness*. San Francisco: W. H. Freeman & Co.

Glynn, S. M. (1991). Explaining science concepts: A teaching-with-analogies model. In S. M. Glynn, R. H. Yeanny, & B. K. Britton (Eds.), *The psychology of learning science* (pp. 219-240). Hillsdale, NJ: Lawrence Erlbaum.

Merritt, S. M., Bruen, C. J., East, J. P., Grantham, D., Rice, C., Proulx, V. K., Segal, G., & Wolf, C. E. (1994). ACM model high school computer science curriculum. *The report of the task force of the pre-college committee of the Education Board of the ACM* (pp. 1-25).

Noss, R., & Hoyles, C. (1996). *Windows on mathematical meanings*. Dordrecht, The Netherlands: Kluwer.

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

Stavy, R., & Tirosh, D. (1993). When analogy is perceived as such. *Journal of Research in Science Teaching*, *30*, 1229-1239.

Tucker, A. B. (Ed.). (1991). Computing curricula 1991, A summary of the ACM/IEEE Joint Curriculum Task Force Report. *Communication of the ACM*, *34*, 69-84.

Direct reprint requests to:

Michal Armoni
Computer Science Department
The Open University of Israel
108 Ravutski St., POB 808
Raanana 43107, Israel
e-mail: michal@openu.ac.il