TEACHING REDUCTIVE THINKING

Michal Armoni
Computer Science Department
The Open University of Israel
108 Ravutski Street
P. O. Box 808
Raanana, 43107 Israel
and
Constantiner School of Education
Tel-Aviv University
Box 39040
Tel-Aviv, 69978 Israel
michal@openu.ac.il

Judith Gal-Ezer
Computer Science Department
The Open University of Israel
108 Ravutski Street
P. O. Box 808
Raanana, 43107 Israel

INTRODUCTION

When dealing with a complex problem, solving it by reduction to simpler problems, or problems for which the solution is already known, is a common method in mathematics and other scientific disciplines, as in computer science and, specifically, in the field of computability. However, when teaching computational models (as part of computability) this topic is not usually explicitly emphasized. The syllabus of most courses dealing with computational models includes closure properties of regular and context-free languages, properties which utilize the theoretical basis for using reduction to prove that given languages are regular or context-free. Indeed, such proof problems are usually given in any computational models course, and solved using closure properties. However, there is usually no didactic explicit emphasis on the reductive nature of such solutions, on demonstrating the advantages of using reduction for solving such problems or on the effect of the nature of reduction on the characteristics of the solution. We believe that developing reductive thinking patterns is an important goal in any scientific discipline, and specifically in Computer Science (CS). Problems that deal with computational models can nicely serve to demonstrate and enable practice of such thinking patterns, especially if these problems are carefully chosen, and have a few possible reductive solutions. This paper demonstrates this principle using a number of problems for two computational models: The finite automata model (representing finite state machines with no additional memory) and the pushdown automata model (representing finite state machines equipped with an infinite memory, which may be accessed according to the Last-In-First-Out principle).

These problems were used, together with other problems, in a wider study, in which the authors examined whether high school students tend to use reduction while solving problems dealing with computational models, and the characteristics of their solutions [1]. Even though the topic of computational models is considered a difficult and abstract topic, the designers of the new Israeli CS high school curriculum decided that it was important to expose high school CS students to the theoretical foundations of the discipline, and to the abstract thinking patterns characterizing it [2, 3]. We view reductive thinking as one of these patterns. The same didactic principle – using appropriate problems with a few possible reductive solutions – can also be applied in college and university courses dealing with computational models.

FINITE AUTOMATA

For the finite automata model we present two problems. The first one is simpler, but it clearly demonstrates that even a simple problem can have a rich variety of possible solutions. It also shows an important principle: in reductive solutions, the nature of the reduction usually affects the technical complexity of the solution. The second problem is a bit more complicated and it serves to demonstrate that the above-mentioned principle can be taken to an extreme, at which point it becomes impossible to apply it.

A Simple Problem: Let L be the language over the alphabet $\{a, b, c\}$, such that each word in L takes the form $a^n b^m c^k$ $(n, m, k > 0)$, and in each word either the number of a's is even or the number of c's is at least 3. Is L regular? Prove your claim.

Although L does not seem very complex, this problem has many possible solutions. The regularity of L can be proved directly, by designing a finite automaton that accepts it. Such an automaton is quite complex: A minimal automaton, with 9 states, would be deterministic (since non-determinism does not help in this case) but would be non-complete, utilizing the freedom of omitting transitions. The problem can also be solved using reduction: L can be decomposed into simpler base languages and then an automaton can be constructed for each, thus proving that they are all regular. At this stage, there are two possible solution methods: Either by using closure properties of regular languages to deduce the regularity of L (without constructing an automaton accepting it), or by using known construction algorithms that compose the automata constructed for the base languages into an automaton for L. In both cases, the technical effort needed to construct the base automata is equal, and depends on the level of decomposition, as will be demonstrated below. However, the second option involves another technical phase, which may be quite long and tedious. The closure properties and algorithms needed for this proof and for the other proofs in this paper can be found in many textbooks about automata and formal languages, such as [4].

We now present the possible ways of decomposition and demonstrate that the finer the decomposition, the simpler the technical complexity of the solution. For simplicity, all the base languages are defined over the alphabet $\{a, b, c\}$. These decompositions use union, intersection or concatenation, and regular languages are closed under these three operations. It is obvious that before presenting these problems or other more complex problems, the students have already covered the material relating to closure properties.

- L can be decomposed into two regular base languages: $L_1 = \{a^{2n}b^m c^k \mid n,m,k > 0\}$, $L_2 = \{a^n b^m c^k \mid n,m > 0, k \geq 3\}$. $L = L_1 \cup L_2$. The corresponding automata are not simple. The first has 5 states, and the second has 6.
- L can be decomposed into three regular base languages: $L_3 = \{a^n b^m c^k \mid n,m,k > 0\}$, $L_4 = \{w \mid \text{the number of } a\text{'s in } w \text{ is even}\}$, $L_5 = \{w \mid w \text{ contains at least 3 } c\text{'s}\}$. $L = L_3 \cap (L_4 \cup L_5)$. The corresponding automata are quite simple and standard. The first one has 4 states, the second has 2 states and the third has 4 states.
- L can be decomposed into four regular base languages: L_4 and L_5, as defined above, and L_3 decomposed into $L_6 = \{a^n b^m \mid n,m > 0\}$ and $L_7 = \{c^k \mid k > 0\}$ (or symmetrically, $\{a^n \mid n > 0\}$ and $\{b^m c^k \mid m,k > 0\}$). $L = (L_6 \cdot L_7) \cap (L_4 \cup L_5)$. The corresponding automata for L_6 and L_7 are very simple (having 3 and 2 states, respectively). This solution may seem a bit artificial, but it was suggested by quite a few high school students.
- Another decomposition into four regular base languages is the following: L_6 and L_7, as defined above, $L_8 = \{a^{2n} b^m \mid n,m > 0\}$ and $L_9 = \{c^k \mid k \geq 3\}$. $L = (L_8 \cdot L_7) \cup (L_6 \cdot L_9)$. Corresponding automata for L_8 and L_9 have 4 states each.
- L can be decomposed into five very simple regular base languages: L_7 and L_9 as defined above, $L_{10} = \{a^n \mid n > 0\}$, $L_{11} = \{a^{2n} \mid n > 0\}$,

$L_{12} = \{b^m \mid m > 0\}$. $L = (L_{10} \cdot L_{12} \cdot L_9) \cup (L_{11} \cdot L_{12} \cdot L_7)$. The corresponding automata have 2, 4, 2, 3 and 2 states, respectively. Three of these automata (for L_7, L_{10} and L_{12}) are isomorphic to each other, and differ only in the letters appearing on the transitions, what further diminishes the technical complexity of the solution since actually only 3 simple automata are needed.

- And finally, there is another solution in which L is decomposed into five even simpler regular base languages: L_4, L_5, L_7, L_{10} and L_{12}, all defined above. $L = (L_{10} \cdot L_{12} \cdot L_7) \cap (L_4 \cup L_5)$. The corresponding automata have 2, 4, 2, 2 and 2 states respectively, and again, the last three automata are isomorphic to each other.

This simple example demonstrates clearly that finer decomposition leads to solutions which are technically simpler. This principle is usually true, and it will be demonstrated again in the second example.

A Complex Problem: Let L' be the language over the alphabet $\{a, b, c\}$ which contains all the words, exactly, for which the following condition holds: 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. Is L' regular? Prove your claim.

A direct solution for this problem involves constructing an automaton with 16 or 17 states (non-deterministic or deterministic automaton, respectively). A corresponding non-deterministic finite automaton is given in Figure 1. Once this automaton is designed, it is not difficult to see a recursive pattern, but for high school students, designing this automata is not a trivial assignment, and many failed to realize in advance that this automata can be designed recursively (or iteratively, from a different point of view).

We will give three additional reductive solutions which use decomposition into regular base languages over the alphabet $\{a, b, c\}$. These decompositions use union, intersection or concatenation, and the regular languages are closed under these three operations.

- L' can be decomposed into four base languages: L_1 contains all the words which do not include a and do not include b. L_2 contains all the words which have exactly one a and one b. L_3 contains all the words with exactly two a's and two b's. L_4 contains all the words with exactly three a's and three b's. L' is the union of these four languages. The corresponding automata are given in Figures 2-5.

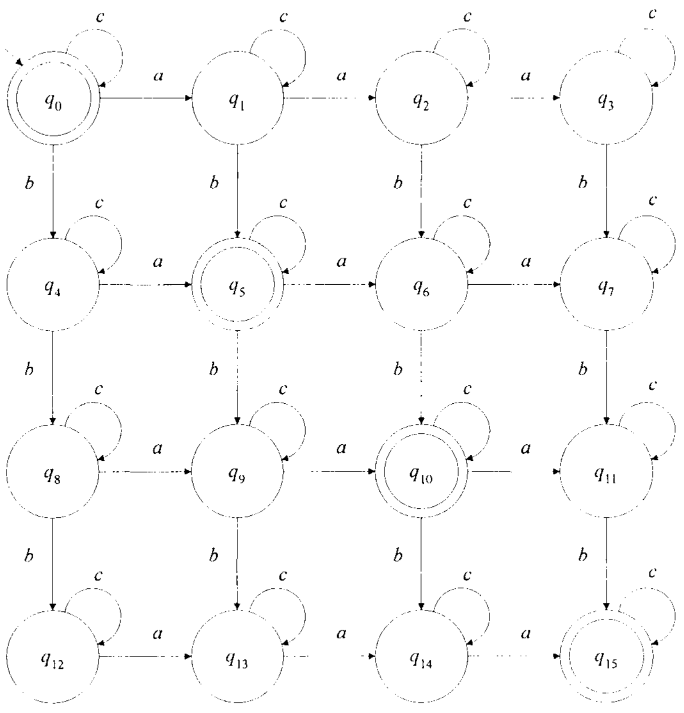

Figure 1: Automaton for L'

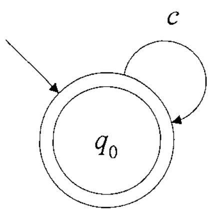

Figure 2: Automaton for L_1

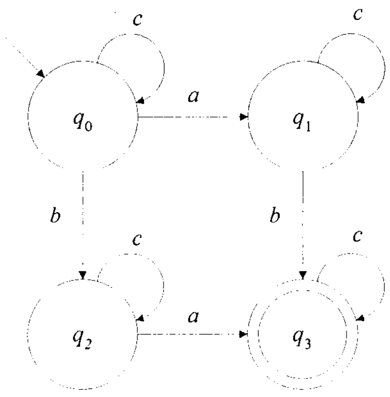

Figure 3: Automaton for L_2

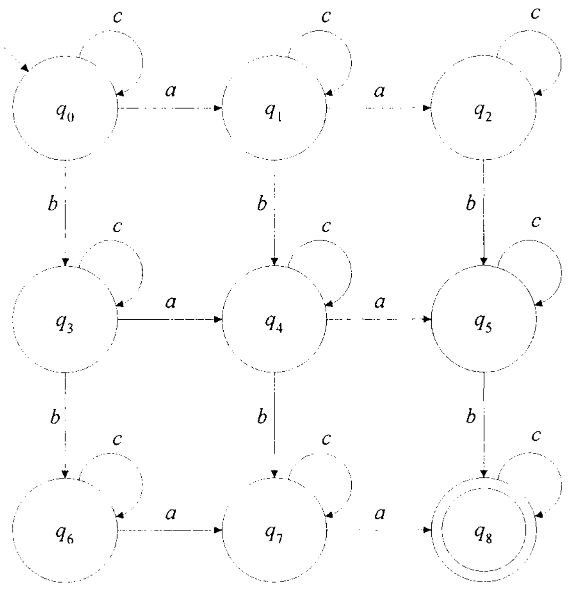

Figure 4: Automaton for L_3

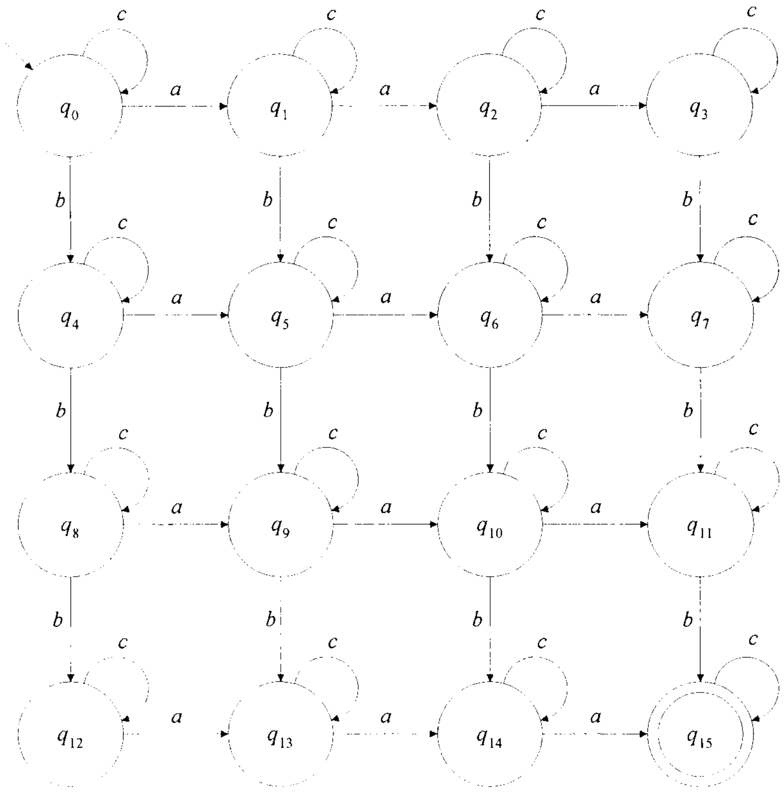

Figure 5: Automaton for L_4

The technical complexity of this solution is not much simpler than that of the direct one, since the automaton for L_4 is exactly the same size as the direct automaton for L' (The only difference is in the set of final states). However, this solution has one advantage: It shows the recursive pattern, mentioned above. That is, in a way, the four automata can be derived from each other, thus disintegrating the construction of the last automaton into simpler sub-phases. The first and second automata are very simple. After successfully constructing the third automaton, it is easier to understand how to use the same principle, in order to construct the fourth one.

- L' can be decomposed into seven regular base languages: L_1 is defined above (it contains all the words which do not include a and do not include b). L_5 contains all the words which include exactly one a. L_6 contains all the words with exactly one b. L_7 contains all the words with exactly two a's. L_8 contains all the words with exactly two b's. L_9 contains all the words with exactly three a's. L_{10} contains all the words with exactly three b's. Now, $L' = L_1 \cup (L_5 \cap L_6) \cup (L_7 \cap L_8) \cup (L_9 \cap L_{10})$.

The technical complexity of this solution is very slight. An automaton for L_1 (given in Figure 2, above) is not difficult. Automata for L_5, L_7 and L_9 are similar: The automaton for L_7 can be derived from that for L_5, by adding one state, and similarly, the automaton for L_9 can be derived from that for L_7, by adding one state. The automata for L_6, L_8 and L_{10} can be derived from the automata for L_5, L_7 and L_9, respectively, by replacing letters on the transitions. All six automata are given in Figures 6 to 11.

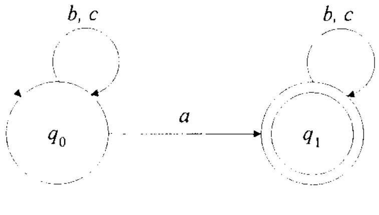

Figure 6: Automaton for L_5

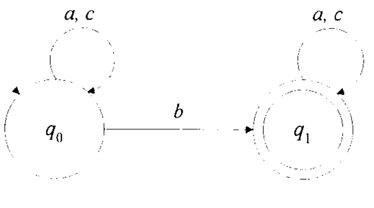

Figure 7: Automaton for L_6

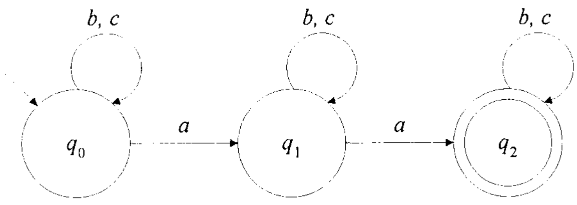

Figure 8: Automaton for L_7

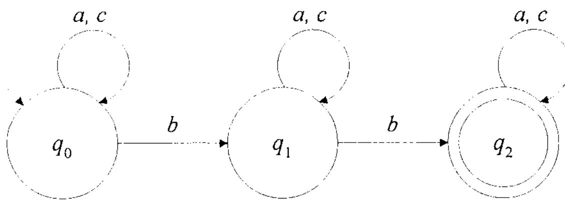

Figure 9: Automaton for L_8

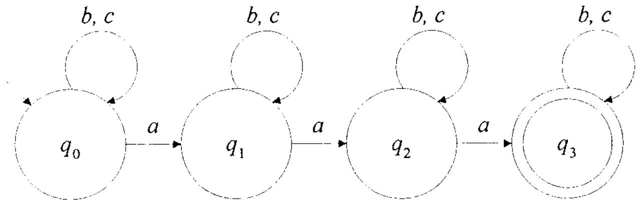

Figure 10: Automaton for L_9

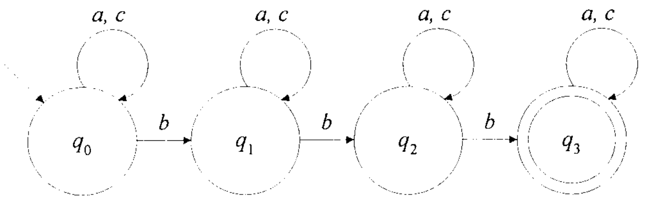

Figure 11: Automaton for L_{10}

• The third solution demonstrates – to an extreme – how finer decomposition simplifies the technical complexity. However, in this case, the decomposition is so fine, that this solution seems too complicated. This decomposition uses only three very simple base languages: L_1, defined above (containing all the words which do not include *a* and do not include *b*). L_{11} contains all the words with exactly one *a* and no *b*'s. L_{12} contains all the words with exactly one *b* and no *a*'s. Corresponding automata for these languages (in Figure 2, above, and in Figures 12 and 13) have 2 states each.

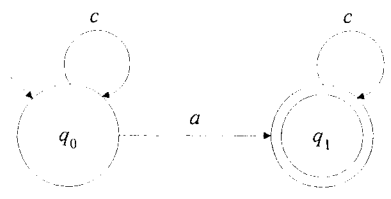

Figure 12: Automaton for L_{11}

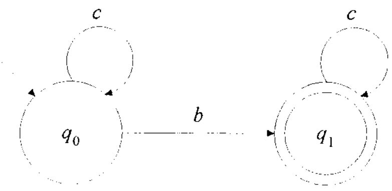

Figure 13: Automaton for L_{12}

Using these base languages L' can be defined as:

$$\begin{aligned}
&L_1 \cup (L_{11}\cdot L_{12}) \cup (L_{12}\cdot L_{11}) \cup (L_{11}\cdot L_{11}\cdot L_{12}\cdot L_{12}) \cup \\
&(L_{11}\cdot L_{12}\cdot L_{11}\cdot L_{12}) \cup (L_{11}\cdot L_{12}\cdot L_{12}\cdot L_{11}) \cup \\
&(L_{12}\cdot L_{11}\cdot L_{11}\cdot L_{12}) \cup (L_{12}\cdot L_{11}\cdot L_{12}\cdot L_{11}) \cup (L_{12}\cdot L_{12}\cdot L_{11}\cdot L_{11}) \cup \\
&(L_{11}\cdot L_{11}\cdot L_{11}\cdot L_{12}\cdot L_{12}\cdot L_{12}) \cup (L_{11}\cdot L_{11}\cdot L_{12}\cdot L_{11}\cdot L_{12}\cdot L_{12}) \cup \\
&(L_{11}\cdot L_{11}\cdot L_{12}\cdot L_{12}\cdot L_{11}\cdot L_{12}) \cup (L_{11}\cdot L_{11}\cdot L_{12}\cdot L_{12}\cdot L_{12}\cdot L_{11}) \cup \\
&(L_{11}\cdot L_{12}\cdot L_{11}\cdot L_{11}\cdot L_{12}\cdot L_{12}) \cup (L_{11}\cdot L_{12}\cdot L_{11}\cdot L_{12}\cdot L_{11}\cdot L_{12}) \cup \\
&(L_{11}\cdot L_{12}\cdot L_{11}\cdot L_{12}\cdot L_{12}\cdot L_{11}) \cup (L_{12}\cdot L_{11}\cdot L_{11}\cdot L_{11}\cdot L_{12}\cdot L_{12}) \cup \\
&(L_{12}\cdot L_{11}\cdot L_{11}\cdot L_{12}\cdot L_{11}\cdot L_{12}) \cup (L_{12}\cdot L_{11}\cdot L_{11}\cdot L_{12}\cdot L_{12}\cdot L_{11}) \cup \\
&(L_{11}\cdot L_{12}\cdot L_{12}\cdot L_{11}\cdot L_{11}\cdot L_{12}) \cup (L_{11}\cdot L_{12}\cdot L_{12}\cdot L_{11}\cdot L_{12}\cdot L_{11}) \cup \\
&(L_{11}\cdot L_{12}\cdot L_{12}\cdot L_{12}\cdot L_{11}\cdot L_{11}) \cup (L_{12}\cdot L_{11}\cdot L_{12}\cdot L_{11}\cdot L_{11}\cdot L_{12}) \cup \\
&(L_{12}\cdot L_{11}\cdot L_{12}\cdot L_{11}\cdot L_{12}\cdot L_{11}) \cup (L_{12}\cdot L_{11}\cdot L_{12}\cdot L_{12}\cdot L_{11}\cdot L_{11}) \cup \\
&(L_{12}\cdot L_{12}\cdot L_{11}\cdot L_{11}\cdot L_{11}\cdot L_{12}) \cup (L_{12}\cdot L_{12}\cdot L_{11}\cdot L_{11}\cdot L_{12}\cdot L_{11}) \cup \\
&(L_{12}\cdot L_{12}\cdot L_{11}\cdot L_{12}\cdot L_{11}\cdot L_{11}) \cup (L_{12}\cdot L_{12}\cdot L_{12}\cdot L_{11}\cdot L_{11}\cdot L_{11}).
\end{aligned}$$

We thus see there is a trade-off – technical complexity vs. the complexity of the decomposition. In a way, the overall complexity of the solution depends on the balance point between them.

PUSHDOWN AUTOMATA

Pushdown automata are usually quite complicated and difficult to design, since their transition function ranges over a tripled Cartesian product. Therefore, simplifying the technical complexity of the solutions in this case, using reduction, can be significant. Consider the following problem, which may help demonstrate the advantages of reductive solutions in this model:

Prove that the language $L'' = \{a^n b^m c^k \mid n,k \geq 0, m = n+r, r \neq k, r \geq 0\}$ over the alphabet $\{a, b, c\}$ is context free.

As was the case for the first two examples, one optional solution is the direct one: constructing a pushdown automaton that accepts L'', thus proving it is context free. However, this isn't a simple automaton: This automaton (accepting by final state) should have six phases. In the first phase it reads a's, while pushing A into the stack for each a it reads. The next phase is reading b's. In this phase it pops an A from the stack for each b it reads. When it reaches the bottom of the stack (that is, the number of b's read is equal to the number of a's) it starts the third phase. In this phase it reads b's, while pushing a B into the stack for each b it reads. In this phase the automaton should be in a final state (since $r>k=0$), and a double bottom for the stack should be created by a special letter, at the beginning of the phase. The fourth phase is reading c's. In this phase the automaton pops a B from the stack for each c it reads, while remaining in a final state (since $r>k>0$), until it reaches the special letter which marks the double bottom of the stack. Reading c when this special letter is at the top of the stack causes the automaton to start the fifth phase, in which it moves to a non-final state (since $r=k$) while popping the special letter. Reading another c starts the sixth and last phase, in which the automaton is in a final state (since $r<k$), and can continue reading c's without touching the stack. In addition to the transitions described here, the automaton should also have transitions which handle the special cases ($n=0, r=0, r=1$ or a combination of these).

We will present three other solutions, all reductive and using decomposition into simpler context-free base languages. These decompositions use concatenation and union, and the context-free languages are closed under these two operations.

- L'' can be decomposed into two base languages:

  $L'' = L_1 \cdot L_2$, $L_1 = \{a^n b^n \mid n \geq 0\}$ over the alphabet $\{a, b\}$ and $L_2 = \{b^r c^k \mid r,k \geq 0, r \neq k\}$ over the alphabet $\{b, c\}$. An automaton for L_1 is a classic one, appearing in most text books. An automaton for L_2 can be derived from it, using an additional state, adding a few transitions, and replacing each a for b and each b for c.

- L'' can be decomposed into three base languages:

  $L' = L_1 \cdot (L_3 \cup L_4)$, L_1 as defined above, $L_3 = \{b^r c^k \mid k > r \geq 0\}$ and

$L_4 = \{b^r c^k \mid r > k \geq 0\}$. Automata for L_3 and L_4 can be derived, through quite simple changes, from the automaton for L_1. However, the technical complexity of this solution is not significantly different than that of the previous one.

- L'' can be decomposed into four base languages: L_1 as defined above, $L_5 = \{b^r c^r \mid r \geq 0\}$, $L_6 = \{b^k \mid k > 0\}$ and $L_7 = \{c^k \mid k > 0\}$. $L'' = (L_1 \cdot L_6 \cdot L_5) \cup (L_1 \cdot L_5 \cdot L_7)$. The technical complexity of this solution is very simple: The automaton for L_5 is actually isomorphic to the automaton for L_1, and can be derived from it by replacing each a with b and each b with c. L_6 is a regular language, for which a 2-state finite automaton is not difficult to construct, and an automaton for L_7 can be derived from it, by replacing each b with c.

SUMMARY

While carrying out the study in [1], in which we examined various problems related to the perception of computational models by high school students, we came to the conclusion that developing reductive thinking is important and should be explicitly emphasized when teaching computational models.

In order to develop this skill the explicit use of reductive solutions and the demonstration of the advantages of the reductive strategy is recommended wherever possible throughout the computer science curriculum. In this paper we have shown how this can be done with problems dealing with computational models. Although the problems presented here were used for high school students, appropriate problems can be defined that are suitable for college and university courses. Using such problems demonstrates the trade-off between technical complexity and the complexity of decomposition. In most cases the finer the decomposition, the simpler the technical complexity. Thus, using such problems encourages students to look for finer decompositions, and therefore helps to develop reductive thinking.

REFERENCES

1. Michal Armoni, Judith Gal-Ezer, and Dina Tirosh, "Solving Problems Reductively", Accepted for publication in *Journal of Educational Computing Research*, ISSN: 0735-6331.
2. J. Gal-Ezer, C. Beeri, D. Harel and A. Yehudai, "A high school program in computer

science", *Computer*, Vol. 28, pp. 73-80, ISSN: 0018-9162 (1995).

3. J. Gal-Ezer, and D. Harel, "Curriculum and course syllabi for a high-school program in computer science", *Computer Science Education*, Vol. 9, pp. 114-147, ISSN: 0899-3408 (1999).
4. J. E. Hopcroft, and J. D. Ullman, *Introduction to Automata Theory, Languages and Computations,* Addison-Wesley, Reading, MA, ISBN: 020102988x (1979).