Jl. of Computers in Mathematics and Science Teaching (2006) **25**(4), 325-359

Introducing Nondeterminism

MICHAL ARMONI AND JUDITH GAL-EZER
The Open University of Israel
Israel
michal@openu.ac.il
galezer@cs.openu.ac.il

Nondeterminism is an essential concept in mathematics and one of the important concepts in computer science. It is also among the most abstract ones. Thus, many students find it difficult to cope with. In this article, we describe some didactic considerations, which guided the development of a "Computational Models" course for high school students, a course in which the concept of nondeterminism is introduced. Some of these considerations are relevant to college and university students as well. We also discuss students' perceptions of nondeterminism and their achievements in this area. Our findings show that many students prefer to avoid nondeterminism, even when it can significantly simplify the solution's design process. We analyze and categorize the students' solutions, thus shedding light on their perceptions of the abstract concept of nondeterminism.

Nondeterminism is an essential concept in computer science as well as in mathematics. In a typical bachelor CS program, the concept is introduced in the computational models course, and is often discussed again later in an artificial intelligence course. Sometimes it is also discussed in the context of operating systems and distributed algorithms. In a traditional high school program, the concept is rarely included. A high school student may, in a sense, be exposed to the concept when studying probability theory. In that context, nondeterminism is forced by the random nature of events.

In the CS high school curriculum described by Gal-Ezer, Beeri, Harel and Yehudai (1995; Gal-Ezer & Harel, 1999) the concept of nondeterminism

is introduced in the Computational Models (CM) unit. The CM unit motivates the need for nondeterministic computational models, and encourages students to choose them, even when deterministic models are sufficient, by demonstrating their advantages.

This unit is unique, compared to other units in this curriculum, and to the best of our knowledge is also unique compared to other high school curricula in other countries. For example, the ACM high school computer science curriculum (Merritt, 1994) includes very few references to few of the topics of the CM unit, only as optional topics, and the more recent ACM K-12 computer science curriculum (Tucker, Deek, Jones, McCowan, Sthepenson, & Verno, 2003) mentions only limits of computability, in one of its five units, as one topic among 10.

In this article we describe the introduction of nondeterminism in the CM unit, and the didactic considerations that guided it, some of which can be applied on college and university levels as well. We also describe the results of a study we conducted on the perception of nondeterminism (partial information on this study was given in Armoni & Gal-Ezer, 2003). This study was part of a wider study, which examined various aspects of the teaching and learning process of the CM unit, such as students' achievements (Armoni & Gal-Ezer, 2004), teaching reductive thinking (Armoni & Gal-Ezer, 2005), and the tendency of students to choose reductive solutions (Armoni, Gal-Ezer, & Tirosh, 2005). The aspect of nondeterminism studied here focuses on the level of use of nondeterminism by students when they have freedom of choice, from which we can deduce their perceptions of this concept.

This article is organized as follows: in the second section " Computational Models in a High School CS Curriculum" we give a brief description of the CM unit; the third section "The Introduction of Nondeterminism" describes the way nondeterminism is introduced; the fourth section "The Study" describes the findings of our study regarding the perception of nondeterminism. Our conclusions and suggestions for further research are discussed in the fifth and last section.

COMPUTATIONAL MODELS IN A HIGH SCHOOL CS CURRICULUM

The CM unit is part of a high school curriculum described in detail in Gal-Ezer et al. (1995) and Gal-Ezer and Harel (1999). According to this curriculum, students who decide to take a comprehensive computer science program, study five 90 hours courses (units), taught over two or three school

years. The last unit is one of the two elective units in this curriculum, and one of the alternatives offered for it is the CM unit. The CM unit is unique, compared to other units of this curriculum, since an important portion of it is dedicated to discussing properties of abstract computational models, rather than implementations within these models. It is the only unit in the curriculum, which does not have a clear aspect of solving algorithmic problems and implementing the solutions in a programming language.

CM introduces finite automata (deterministic and nondeterministic), pushdown automata, and Turing machines. We will elaborate on the choice of these models in the following subsection. 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 main educational goal of the CM unit is to expose students to theoretical thinking patterns, which are mathematical in nature, including abstraction and generalization (demonstrated by defining abstract models, inspired by simple automatic machines), abstract analysis (demonstrated by investigating the introduced computational models and their properties) and abstract conceptual flexibility (demonstrated by various changes in the definitions of models and by investigating the effect of such changes).

To preserve the uniqueness of the CM unit, compared to other units in this high school CS curriculum, it was decided to focus on theoretical issues, and to relate to automata design as a tool exploited in theoretical discussions. Therefore, this unit intentionally does not mention "real life" various applications of computational models, such as compilers, natural language processing, and so forth, nor does it integrate any simulation or visualization tools into the teaching process. We assumed that introducing such tools would only affect the perception of the technical aspects (i.e., designing automata), aspects which we found to be easier to cope with (Armoni & Gal-Ezer, 2004). Also, since such tools are usually quite attractive, and since most high school students are not fully mature, we were afraid they might invest too much time and attention in these tools, disrupting the learning process of the theoretical aspects.

The CM unit was developed by a team chaired by the first author, in consultation with the second. The teaching process is guided by a textbook and a teacher guide especially designed for the unit.

The Choice of Models

To fit into the 90 hours limit, not all classical models, traditionally introduced in a college or university computational models course, could be introduced in the CM unit. A didactic decision had to be taken, regarding the choice of models. The CM unit focuses on models, which are automata-based and does not introduce other common models, such as regular expressions or grammars. This restriction leaves us with relatively "concrete" models, as they resemble real automatic machines, and thus may be easier to cope with for high school students. In addition, these models have very similar definitions, enabling the introduction of each new model as based on the preceding one. This minimizes the conceptual adaptation that a student needs to learn a new model. For example, the Turing machine model is presented as a generalization of pushdown automata, achieved by permitting access to cells beneath the top cell of the stack. In spite of the restriction to automata-based models the chain of models we introduce is rich enough and enables meaningful theoretical discussion, through which major concepts in computer science are introduced: Some models are equivalent and some are not, some models are nondeterministic, some models differ in their closure properties and in the contribution of nondeterminism to their computational power, and one of the models is equivalent to a computer program (or an algorithm).

The CM Unit Syllabus

The CM unit consists of three parts: (a) finite automata, (b) pushdown automata, and (c) Turing machines.

The *first* part introduces three computational models: Deterministic Finite Automata (DFA), Noncomplete Deterministic Finite Automata (NCDFA) and Nondeterministic Finite Automata (NFA). The introduction of each new model is followed by a technical part, which focuses on designing automata within this model, and then there is a theoretical discussion. The theoretical aspects discussed in the first part include the definition of regular languages, computational limits (that is, the existence of nonregular languages), equivalence of the three models and some closure properties.

The *second* part of the CM unit focuses on the Pushdown Automata model (PDA). Again, the technical part is followed by a theoretical discussion in which PDA are proved to be stronger than finite automata, the computational limits of the model are discussed, and some closure properties of

the family of languages accepted by this model (context-free languages) are proved.

The *third* and final part of the CM unit is dedicated to the Turing Machine model (TM). The technical aspects include designing simple machines that accept formal languages, and simple machines that calculate functions. The theoretical aspects include a discussion of the computational power of TM (i.e., the Church-Turing thesis) and, in particular, the equivalence of the new model to a computer; and a discussion of the new model computational limits, demonstrated by proving the noncomputability of the halting problem.

THE INTRODUCTION OF NONDETERMINISM

The concept of nondeterminism is introduced and discussed in the fourth chapter of the CM unit, which takes about 15 hours.

We chose a way somewhat different from the traditional way of introducing nondeterminism. The nondeterministic finite automaton (NFA) model is usually defined as a straightforward version of the deterministic finite automaton (DFA) (Hopcroft & Ullman, 1979). In a DFA, the transition function maps each pair of a state and an input letter to a single state, while in an NFA, the transition function maps each pair of a state and an input letter to a set of states (which may also be empty). Figure 1 shows an NFA, accepting the language that contains all the words over $\{a, b, c\}$ that end with bc.

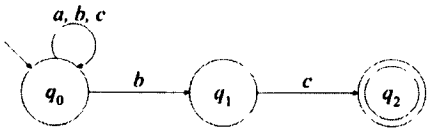

Figure 1. An Example of an NFA

The definition of NFA is a generalization of the definition of DFA, since it enables the range of the transition function to include any sets of states and not just singletons. However, this generalization encapsulates two important differences between a DFA and an NFA, only one of which relates to nondeterminism: The first difference is that an NFA permits omitting transitions (that is, a state and an input letter can be mapped to an empty set), but this option still preserves the deterministic nature of the model. The second difference is that in an NFA, a state and an input letter can be mapped to a set of two or more states, and this introduces real nondeterminism.

We thought that since the concept of nondeterminism (expressed by the second difference) is an abstract one, and is therefore probably difficult to understand, it is better to introduce it by itself, thus making its perception easier. Therefore, the introduction of NFA in the CM unit is preceded by the introduction of another model: the Noncomplete Deterministic Finite Automaton (NCDFA). In this model, the transition function maps each state and input letter to a single state or to an empty set of states. Figure 2 shows an example of an NCDFA, accepting the language of all the words over {*a*, *b*, *c*} that begin with *bc*.

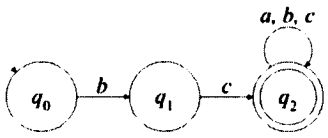

Figure 2. An example of an NCDFA

Thus, the first difference between a DFA and an NFA, which permits omitting transitions while preserving the deterministic nature of the model, is shown in the definition of the NCDFA; while the second difference, which permits nondeterminism, is expressed only in the definition of the next model, the NFA. The abstract concept of nondeterminism is thus isolated, and is introduced by itself. The addition of the NCDFA model also enriches the variety of models introduced in the CM unit, and enables practicing comparison of models.

The desire to focus on nondeterminism guided the decision to make another change in the traditional definition of an NFA. The common definition of NFA permits ε-transitions (transitions that can be made without reading any input symbol). To avoid additional perceptional difficulties, it was decided not to include ε-transitions in the CM unit, for the following reasons:

- Introducing ε-transitions may enhance the expected difficulties in the perception of nondeterminism since ε-transitions increase the level of nondeterminism.
- ε-transitions do not correspond to the familiar role of an automaton as representing machines activated by external events. In a sense, an ε-transition represents a reaction of the system to an internal event, and thus it is expected to be less natural and intuitive.
- ε-transitions do not enhance the computational power of NFAs.
- In accordance with the spirit of the CM unit, a new model cannot be introduced without discussing its properties and comparing it to previously introduced models. The unit time limits could not enable

both introducing ε-transitions and discussing the equivalence of the two variations of NFA.

One of the challenges in introducing NFA was deciding how to justify the introduction of the new model. In the CM unit, the motivation for introducing NFA is explained in three ways:

- Practicing the "theoretical game" which characterizes theoretical study in CS. That is, after a certain mathematical abstraction is defined (in this case DFA), it is interesting to check the theoretical results of generalizing this definition in various ways and determining whether the resulting models are equivalent to the original one, stronger or weaker.
- A few examples, given in the chapter, demonstrate that for certain formal languages, constructing an NFA is simpler and more natural than constructing a DFA. This natural tendency is used in the theoretical game previously mentioned: If we sometimes tend to design an illegal automaton, which does not comply with the model definition, why not try to generalize the definition to make such an automaton legal, and then examine the effects of this generalization?
- Finally, it is shown that by using the nondeterministic model, additional closure properties of regular languages can be proven.

The last two explanations demonstrate the advantages of the nondeterministic model: The relative ease of designing nondeterministic automata and of proving claims using this model. It is reasonable to assume that emphasizing the first of these two advantages may encourage students to use the nondeterministic model when solving design problems, in which a regular language is given and the student is required to design an automaton accepting it.

To clarify how nondeterministic models work, mainly their acceptance mechanism, the CM unit uses the magic coin metaphor, introduced by Harel (1987): Students are told that they should think of a nondeterministic automaton as if it were equipped with a coin which it flips before it decides which transition to choose. However, this is an unbalanced magic coin—it will always tend to accepting paths. That is, each time the automaton has to select one of several possible transitions, flipping the coin will cause it to choose a transition which can lead to a final state, while reading the suffix of the input word (if such a transition exists). Thus, if there is an accepting path for a given input word, the magic coin will guide the automaton to such a path.

Nondeterminism is again discussed in the CM unit when introducing PDA. The definition of this model is obtained by changing the definition of the last model introduced to the students, the NFA model. The only change is adding a stack to NFA. The resulting model is thus nondeterministic. The desire to keep the changes between models as clean as possible guided our decision not to permit ε-transitions in PDAs as well, and to define acceptance in the PDA model as reaching a final state, and not as emptying the stack. It is well known that both definitions of acceptance of PDA are equivalent; introducing only one definition does not decrease the computational power of the resulting model. It can also be proved that the computational power is not decreased by not permitting ε-transitions.

The PDA model is also used to demonstrate different behaviors of different models in relation to nondeterminism: For finite automata, nondeterminism does not increase the computational power, while for PDA it does.

THE STUDY

The first part of this section describes the method and population of the study, the second describes our findings and the third part discusses nondeterminism in the context of PDAs.

Method and Population

Developing the CM unit involved an experiment, during which the unit was taught in selected schools under the close supervision of the development team. The developers provided the teachers who taught in this experiment with a number of questions, each related to a certain subject. The teachers were asked to include these questions on exams held immediately after the students finished studying the corresponding subjects. The teachers were asked to send the students' full answers to these questions to the research team. Among these questions was the following one, included in the exam given immediately after introducing, practicing, and investigating nondeterminism in the context of finite automata. This question served as the research tool for studying the perception of nondeterminism:

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 holds:

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.

Two main factors are involved in the process of solving this problem: the reduction of the problem into sub-problems and the use of nondeterminism. Though these two factors may seem orthogonal to each other, they are not fully independent. For example, if the student chooses to decompose the language into two or three sublanguages, the resulting sublanguages will be quite simple, and therefore constructing a DFA for each will not be overly complicated. However, in this article we limit ourselves only to the factor of nondeterminism. Reduction was part of our wider study, and is discussed in Armoni, Gal-Ezer, and Tirosh (2005).

The research population included 339 students who took an exam which included the above question and their solutions were sent to us. These students studied in 17 classes, in 9 schools, taught by 11 teachers; 244 students were 12th graders, and 95 were 11th graders.

The teachers were also asked to send the developers the students' answers on the final exams. We received 48 solutions for questions from the final exams, to which nondeterminism was relevant.

The Use of Nondeterminism—Findings

Quantitative results. Most of this section is dedicated to a quantitative analysis of the students' solutions to the question presented earlier. Later we will briefly relate to the findings of the final exams.

After reading the full answers of the 339 students to the question and identifying common characteristics, the authors categorized the solutions in relation to the level of use of nondeterminism and divided them into five groups:

- Fully deterministic solutions.
- Solutions in which the students used decomposition to two or three sublanguages. The automata built for these sublanguages were fully deterministic, and only the use of construction algorithms introduced nondeterminism into the process. We categorized this as a

deterministically-based solution, since when independent thinking was required, the deterministic model was used.
- Almost deterministic solutions, with only a few local nondeterministic behaviors.
- Almost nondeterministic solutions, with only a few instances in which the student ignored the freedom of the nondeterministic model and used redundant transitions.
- Fully nondeterministic solutions.

When categorizing the students' answers, we found no solutions which could be defined as "equally deterministic and nondeterministic," that is, solutions in which the portion of nondeterminism is more or less equal to the portion of determinism, and none seem to be more dominant than the other. This is not surprising. It is reasonable to assume that if students do not understand the nondeterministic mechanism, they will not use it (partially or at all), whereas if they understand the mechanism and its advantages, they will try to use it as much as possible.

It should be emphasized that we did not ask the students specifically to construct a nondeterministic automaton, so they had the freedom of choice. It is possible that if the question had been asked differently, students who did not choose the nondeterministic model might have successfully constructed an NFA. However, since the CM unit emphasizes and demonstrates the relative ease of the design process in the nondeterministic model, as compared to the deterministic model, we reasoned that if, in spite of that, the students preferred to use the deterministic model, this may indicate that they did not fully understand the nondeterministic model. That is, the tendency to use the deterministic model is in itself an important indicator of students' level of understanding of the nondeterministic model.

The distribution of the various types of solutions is shown in Figure 3 (originally in Armoni & Gal-Ezer, 2003).

Figure 4 (originally in Armoni & Gal-Ezer, 2003) is a version of Figure 3, but combines the three deterministic or almost deterministic columns, and the two almost nondeterministic and nondeterministic columns.

About half of the students solved this question deterministically, or almost deterministically. No significant statistical differences were found for grade (11^{th} and 12^{th}) or level of mathematics. These findings definitely indicate a strong tendency toward determinism.

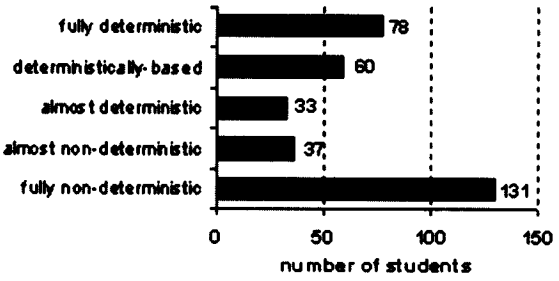

Figure 3. Solutions by type

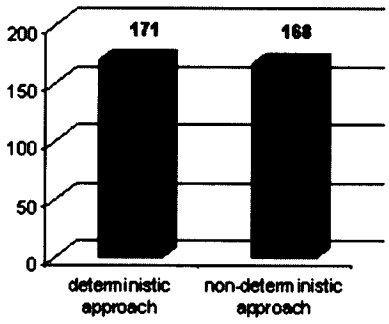

Figure 4. Deterministic versus nondeterministic approach

Our findings for the final exams indicate an even higher tendency to determinism. The final exams were held at the end of the school year, three or four months after the exam including the question above. The final exams included two proof questions for which a nondeterministic approach could result in simpler automata than those induced by a deterministic approach. We will not give here the questions and the complete results, just briefly say that only about a third of the solutions for these questions were nondeterministic or almost nondeterministic. Indeed, the first exams were held immediately after the students learned the nondeterministic model, but they should have been exposed to it throughout the rest of the CM unit. In a proper teaching process, after introducing the nondeterministic model, it should have become the main tool for solving design problems (for regular and context-free languages). Therefore, by the end of the year, students

were expected to feel quite comfortable with nondeterminism and to be fully aware of its advantages. A reasonable explanation for these findings is that those students who did not fully understand the nondeterministic model immediately after it was taught, continued to cling to the deterministic model, and used the nondeterministic approach rarely or not at all. Thus, at the end of the year, they felt even more uncomfortable with the nondeterministic model than they had felt immediately after learning it, since it was not as fresh in their minds.

The teachers were not at the focus of our study. However, we considered the data for the above question for each teacher separately, and found that for about half of the teachers (5 of 11), the ratio between students using the deterministic approach and those using the nondeterministic approach was about 50-50. Regarding the rest of the teachers, for three the ratio was about 60%, 70%, and 85%, in favor of the deterministic approach, while for the other three, the ratio was about 70%, 85%, and 90%, in favor of the nondeterministic approach. In addition, when interviewed, some of the teachers said that they did not feel as comfortable with the nondeterministic model as with the deterministic model, and therefore they did not tend to emphasize the nondeterministic model and its advantages during the teaching process. Thus, there appears to be some preliminary evidence that the teachers themselves vary significantly in their perception of nondeterminism. It therefore seems reasonable to hypothesize that the teacher factor is significant and should be further studied in future research: Is there really a significant variance among teachers regarding their perception of nondeterminism? If there is, is there a clear connection between the teacher's level of perception of nondeterminism and its students' tendency to use nondeterminism?

Qualitative results. While reading the students' solutions, we identified four patterns, which seem to indicate the existence of a problem in the perception of nondeterminism. We believe that these patterns deserve close attention. Teachers should be aware of these patterns, and learn to recognize them in their students' answers in order to identify perception problems. We will briefly describe these patterns and then demonstrate and analyze each through students' solutions.

- The first pattern, **local nondeterminism,** was found among the solutions in the category "almost deterministic." These solutions include automata which are basically deterministic, but contain local nondeterminism expressed in a few nondeterministic transitions. This pattern was found in about 10% of the 339 solutions.

- The second pattern, **locally deterministic solutions,** is "symmetric" to the first. It can be found among the solutions in the category "almost nondeterministic." These solutions include automata which are basically nondeterministic but contain a few transitions characteristic of deterministic automata. This pattern was found in about 11% of the 339 solutions.
- The third pattern—**shifting from nondeterminism to determinism:** In some cases, we could identify traces of the solution process in a student's answer. Sometimes students wrote a few preliminary versions which they chose not to complete. In the few such cases we encountered in our data, the process indicates a change from a nondeterministic model to a deterministic one. Counting the number of these solutions is meaningless since the fact that other students didn't leave such traces does not necessarily mean that they did not go through a similar process.
- The fourth and last pattern, **noncomplete deterministic solutions**, was found among the fully deterministic solutions, or the solutions in which some of the automata for the sublanguages were deterministic. In these cases, the students constructed NCDFAs. Even though the automata constructed in this pattern were fully deterministic, this pattern points at partial understanding of the nondeterministic mechanism. The solutions matching this pattern were few (about 4%).

Each of these four patterns will now be demonstrated using examples of students' solutions.

First Pattern: Local Nondeterminism

The four examples below show basically deterministic solutions, with local nondeterminism.

Example 1 – Direct solution with a hidden decomposition to three sublanguages

Andy designed the automaton shown in Figure 5. This is a direct solution, bur there are clear traces of decomposition: This automaton consists of three basically deterministic segments: $\{q_0, q_1, q_2\}$, $\{q_2, q_3, q_4\}$, $\{q_0, q_5, q_6\}$, accepting the languages of all words containing *ba*, of all words containing *ab*, and of all words ending with *bc*, respectively.

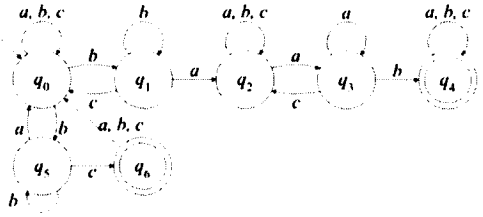

Figure 5. Andy's design

There is local nondeterminism in this automaton in two places: in q_0 with *b* and in q_2 with *a*. Since most of the automaton is fully deterministic, the nondeterminism in these two states is redundant. Perhaps, even though there was no implicit decomposition, Andy designed this automaton by combining segments corresponding to base languages, and this process led to the nondeterminism in q_2 (where two segments are glued), since as a final state in an automaton which accepts the language of all words containing *ba*, q_2 should have a self loop with all the letters of the alphabet.

In q_0 there are three transitions with *b*: One is to q_1, searching for *ba*, the second one is to q_5, searching for *bc*. These two transitions are essential, and this local nondeterminism is probably also due to combining two segments into an automaton accepting a union language. But the self loop with *b* in q_0 is hard to explain. This transition was not made by any combination algorithm, and it is not consistent with the logical structure of this automaton. It is a redundant and artificial nondeterministic addition. Indeed, it does not violate the correctness of this automaton (which still accepts the given language), but it probably points at only partial understanding of the nondeterministic mechanism.

Note that the transition with *b* in q_6 is incorrect. Instead of going back to q_5, it returns to q_0. This is a relatively common mistake when designing automata to look for a string which ends a word. In this specific case, the redundant transition in the initial state covers this mistake, and the automaton remains correct, but we cannot assume that this was the reason for adding this transition.

Example 2 – Full decomposition to three sublanguages

For each of the three sublanguages, Bonnie designed an automaton which is basically fully deterministic (Figures 6-8). All three automata have a self loop with all the letters in the alphabet in their initial state, and this self loop introduces local and redundant nondeterminism. As in the first example, these redundant transitions may indicate a problem in the perception of the concept of nondeterminism. As was the case for Andy, the local non-

determinism in the first automaton (Figure 6) covers up incorrect transitions in the final state.

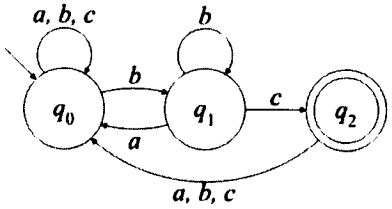

Figure 6. Bonnie's design, part 1

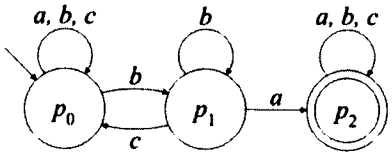

Figure 7. Bonnie's design, part 2

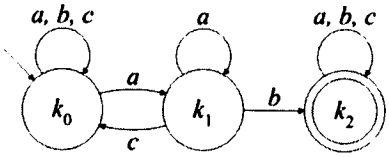

Figure 8. Bonnie's design, part 3

Then Bonnie combined the last two automata (Figures 7 and 8), using the construction algorithm for concatenation, thus introducing additional nondeterministic transitions. In order to construct an automaton for the union language, she did not use the relatively simple algorithm for NFAs but decided to use the Cartesian product algorithm, though it was defined only for deterministic automata and results in a very complicated automaton.

Example 3 – Direct solution

Cal designed an essentially deterministic automaton (Figure 9). There is local nondeterminism in this automaton in the initial state with the letter *b*, and again it is a meaningless and redundant nondeterminism. In this case as well, this redundant transition covers up a mistake—a missing transition in q_1 with the letter *b*. This transition was probably left out by mistake—in q_4 and q_6 which have the same logical role as q_1, the corresponding transition is not missing. Thus, this flaw may be categorized as a merely "syntactic" mistake.

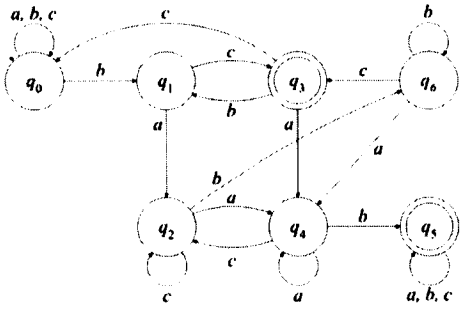

Figure 9. Cal's design

There is another, more significant mistake in Cal's solution: The automaton reaches q_3 after reading *bc*, whether *ba* was found before or not. For example, the word *bcab* is accepted by this automaton though it should be rejected. This mistake probably stems from incorrect logical design, and not from incorrect perception of the nondeterministic mechanism, but it demonstrates that directly designing deterministic automata is complicated, and nondeterminism might help to overcome this complexity.

Interestingly enough, Cal first tried a reductive and nondeterministic solution, decomposing the language into two sublanguages and designing an NFA for each of them. Instead of using the known, simple algorithm to combine these two NFAs into a new NFA, accepting the union language, he turned to a direct, almost deterministic design. Therefore this example fits also the third pattern, of shifting from nondeterminism to determinism.

Example 4 – Direct solution with a hidden decomposition to three sublanguages

Dan's solution (Figure 10) is also a direct solution, with clear traces of decomposition. As in the first example, this automaton consists of three fully deterministic segments, $\{q_0, q_1, q_2\}$, $\{q_2, q_3, q_4\}$ and $\{q_0, q_5, q_6\}$, which are basically deterministic automata accepting the languages of all words containing *ba*, of all words containing *ab*, and of all words ending with *bc*, respectively.

In Dan's solution we can see the familiar mistake—the transitions going back from q_6 miss their correct destinations, and in this case we can again see artificial, redundant local nondeterminism in the initial state, introduced by the self loop, which covers up the incorrect transitions, resulting in a correct automaton. What about the other nondeterminism in the initial state, introduced by the transition with *b* to q_1 and the transition with *b* to q_5? It appears that this nondeterminism enables the automaton to "guess" whether

the input word fits the first condition or the second one. Therefore, the upper segment and the lower segment should be disjoint. However, in spite of this nondeterminism in the initial state, Dan added transitions which connect these two segments and try to fix an incorrect guess, though this is not necessary in a nondeterministic model. These connecting transitions point at only partial understanding of the concept of nondeterminism.

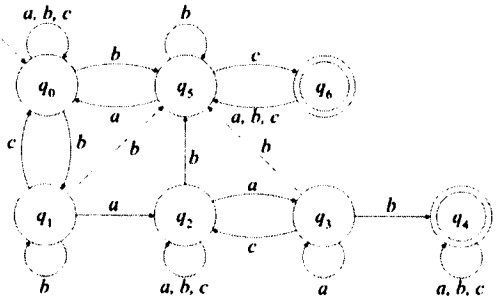

Figure 10. Dan's design

All four examples include redundant nondeterministic transitions, which do not violate the correctness of the automata but are meaningless when added to basically deterministic automata. The existence of such transitions seems to indicate an only partial understanding of the concept of nondeterminism. In all the examples, among the redundant transitions, there was a redundant nondeterministic self loop in the initial state. This was common to most of the other solutions which fit the first pattern. It seems that students who do not fully understand the nondeterministic mechanism identify nondeterminism with a self loop for all the letters in the alphabet in the initial state. Indeed, many of the nondeterministic automata presented in the CM textbook have such a self loop. This may call for a revision of the textbook—adding and emphasizing examples in which nondeterminism is expressed differently.

Second Pattern: Locally Deterministic Solutions

The three examples below represent an essentially nondeterministic approach, but cling to some deterministic characteristics.

Examples 1 and 2 – Full decomposition to three sublanguages

Edith decomposed the given language into three sublanguages. The au-

tomaton accepting the language of all words ending with *bc* was a correct fully-deterministic automaton. Figures 11 and 12 show the automata she designed to accept all the words which contain *ba*, and all the words which contain *ab*, respectively.

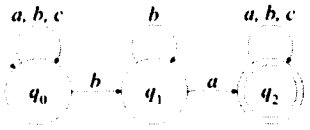

Figure 11. Edith's design, part 1

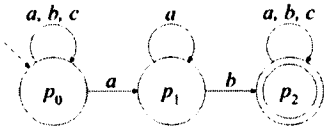

Figure 12. Edith's design, part 2

Fred also decomposed the language into three sublanguages. The automaton accepting the language of all words containing *ba*, and the one accepting the language of all words containing *ab* were correct fully-nondeterministic automata. The automaton Fred designed for accepting all words ending with *bc* is shown in Figure 13.

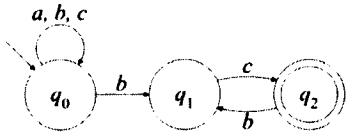

Figure 13. Fred's design

All three automata accept the corresponding languages, and are basically nondeterministic, with one redundant transition (a self loop in the second state of Figures 11 and 12, and a transition with *b* from q_2 in Figure 13). Such transitions are essential for deterministic automata accepting these languages but are meaningless in a nondeterministic context.

Example 3 – Direct solution

George designed the automaton shown in Figure 14. This is a direct and essentially nondeterministic solution, and the resulting automaton indeed accepts the required language. However, the states q_7 and q_8 and the transitions connecting them to the automaton are redundant, due to the nondeterministic nature of this automaton.

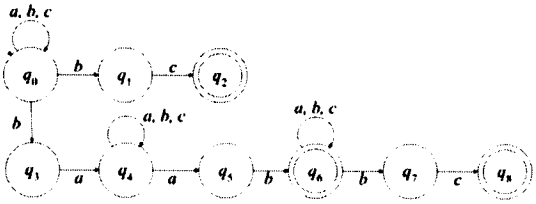

Figure 14. George's design

All three examples demonstrate almost nondeterministic automata with a few redundant transitions (one in the first two examples and two in the last one). It is reasonable to assume that these students' level of perception of nondeterminism is quite good, but not full. The redundant transitions seem to demonstrate residual deterministic thinking, expressed only locally. At some point, these students returned to a deterministic thinking pattern, according to which unsuccessful guesses should be corrected.

Third Pattern: Shifting from Nondeterminism to Determinism

The examples presented for this category represent the few solutions in which we could find traces of the solution process. At the beginning of this process, some or all early versions are nondeterministic or almost nondeterministic (or at least have a meaningful nondeterministic portion), but as the students "improve on" the solution, they construct almost deterministic or fully deterministic automata. In some of these cases, the preliminary versions were indeed incorrect, and usually only simple and local corrections were necessary.

Example 1 – Shifting during a direct solution

In his first attempt, Henry tried a reductive solution, decomposing the language into two sublanguages. The automaton for the first language was fully deterministic, and the one for the second sublanguage was merely a skeleton, with many transitions missing (and with some mistakes). Henry decided not to complete this line, but turned to a direct approach, trying to design a fully deterministic automaton (Figure 15).

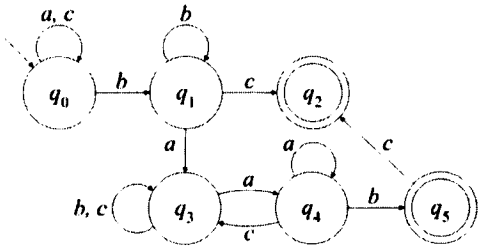

Figure 15. Henry's second attempt

This attempt was also not completed. Henry may have faced some logical problems, such as how to handle a *b* that was read after *ba* had been read. The redundant transition with *b* from q_5 to q_2 also points to a flaw in the logical design of this automaton. Next, Henry designed a basically deterministic automaton (Figure 16), but with meaningful nondeterminism in q_3. Introducing such nondeterminism is the proper way to handle the logical problem of reading *b* after *ba*, mentioned above. Yet, even though this solution is basically correct (he only needed to make q_5 a final state and add a self loop in q_5 with all the letters of the alphabet), Henry decided not to complete it. He left the design unfinished and began another one.

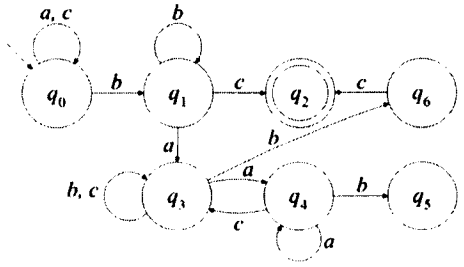

Figure 16. Henry's third design

At this point, Henry abandoned nondeterminism and retreated to the deterministic approach. He designed a fully deterministic automaton (Figure 17), which is almost correct, except for one missing essential transition (a self loop with *b* in q_6).

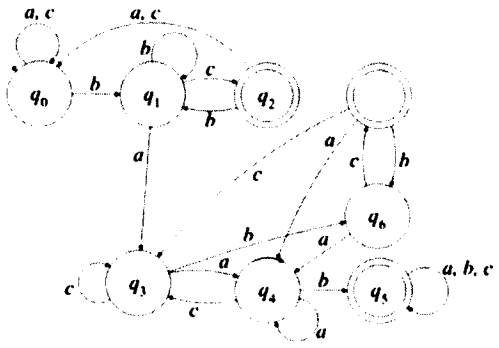

Figure 17. Henry's final design

Example 2 – Shifting during a reductive solution

Ian decomposed the language into two sublanguages. For one sublanguage, he designed a basically nondeterministic automaton, with a redundant self loop with *b* in q_1 (Figure 18). For the second sublanguage, he designed a fully nondeterministic automaton (Figure 19).

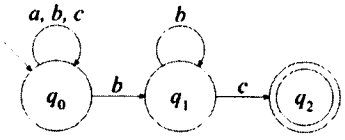

Figure 18. Ian's design, part 1

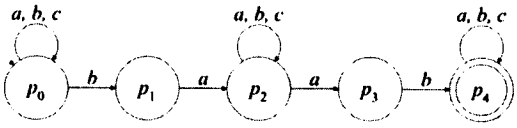

Figure 19. Ian's design, part 2

Next, he combined these two automata into one automaton, accepting the union language (Figure 20). For that purpose he did not use the common algorithm. He joined the first two states of these automata. Then he added a new state, t_6, with a self loop with all the letters of the alphabet, a transition from t_5 to t_6 with all the letters of the alphabet and a self loop with *a* in t_3. All these did not exist in the original base automata and they are redundant.

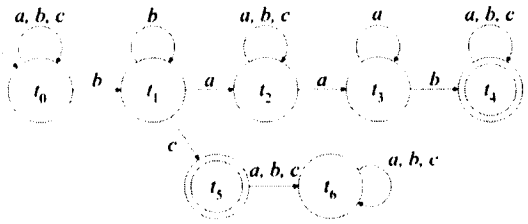

Figure 20. Ian's full design

In both examples, at some point the students designed an automaton with a meaningful portion of nondeterminism, which was correct or almost correct. Yet, both of them preferred to replace it with a deterministic automaton. In Henry's case this was done explicitly, by not completing the nondeterministic version and starting a new deterministic one. Perhaps, after recognizing a problem in the automaton, Henry preferred to shift—or retreat—to the deterministic, not necessarily simpler but perhaps more familiar, model, instead of correcting the mistake within the nondeterministic model. In the second example, Ian's solution, this was done implicitly, by fixing the combination process: Ian added a new state and new transitions, all redundant, and none exists in the original base automata. Adding them implies that Ian felt insecure as to the ability of the nondeterministic model to cope with incorrect guesses. It seems that he thought that the combination process (which added nondeterminism) resulted in an incorrect automaton, which should be fixed. In example 3 of the first pattern (which fits the third pattern as well), the retreat occurred after designing two nondeterministic base automata, before combining them into one automaton.

We found very few solutions in which the solution process could be traced. However, it is reasonable to assume that other students also underwent a similar process—explicitly or implicitly—but did not document it. For example, if we take another look at example 3 of the second pattern (Figure 14), one may speculate that this student may have started with two or three nondeterministic automata, and while combining them added the additional two states, creating some dependency between the two segments which under a nondeterministic approach should be disjoint, thus demonstrating a partial retreat from nondeterminism to determinism.

Fourth Pattern: Noncomplete Deterministic Solutions

The examples presented under this category represent solutions which do not use nondeterminism at all, but instead use NCDFAs. For the languages corresponding to this problem, introducing incompleteness necessitates introducing nondeterminism as well, and thus a noncomplete deterministic solution is necessarily incorrect (the resulting automata do not accept the required language). Below are two examples which fit this pattern.

Example 1 – Decomposition into two sublanguages

Jane decomposed the language into two sublanguages. Neither one of the two corresponding automata (Figures 21 and 22) uses nondeterminism (even in the initial state), but they are both incomplete. The result is incorrect in both cases. For these languages, it is not possible to omit transitions without taking advantage of the freedom offered by nondeterminism. Note that in the second automaton (Figure 22), the right segment is actually fully deterministic (only a self loop with all the alphabet letters is missing in the final state).

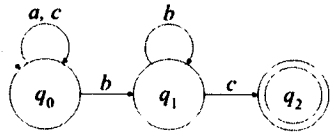

Figure 21. Jane's design, part 1

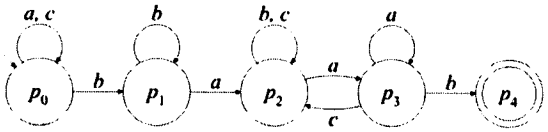

Figure 22. Jane's design, part 2

Example 2 – A direct solution

The skeleton of the automaton designed by Larry (Figure 23) is very similar in its structure to the natural nondeterministic automaton for this language, only this automaton is deterministic. Omitting the nondeterministic self loops in q_0 and q_4 violates the correctness of the automaton (as does also the missing self loop with all the letters of the alphabet in q_6).

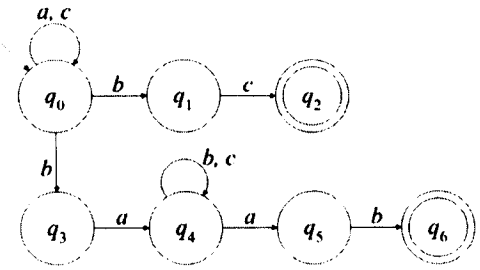

Figure 23. Larry's design

Interestingly enough, in these two examples we see a tendency opposite to the one demonstrated in the first pattern. There we saw redundant self loops that introduced redundant nondeterminism. Here, the self loops are missing, but since the automata are deterministic, omitting these loops violates the correctness of the resulting automata.

A possible explanation for using the freedom of omitting necessary transitions, characteristic of the nondeterministic model, without introducing the nondeterministic transitions that enable it, is that some students find it hard to cope with nondeterministic models, but are willing to "accept" noncomplete deterministic automata. We will relate to this below, when analyzing an interview with one of the students. Thus, even though the automata constructed in this pattern were fully deterministic, the error probably stemmed from a partial understanding of the nondeterministic mechanism. It is important for teachers to understand that such erroneous automata, although deterministic, may indicate a problem in the perception of nondeterminism.

Interviews. We conducted interviews with four students who had finished studying the fourth chapter of the CM unit a few weeks before, and had been tested on the material a week before. Through these interviews we hoped to gain some insight into the solution process, and the reasons for choosing one model over another. Students on various levels of achievement were chosen by the teacher, who did not know in advance what the students would be asked. The four students were asked to solve the question.

After completing their first version of the solution, they were asked about decisions they made while solving the problem. Three of the students gave a nondeterministic solution (two performed a direct construction and one decomposed the language into two sublanguages). One of them was not very cooperative and we were unable to glean any information regarding his decision to use the nondeterministic model. The other two students ex-

plained their choice of model by saying that they thought it was impossible to construct a deterministic automaton for this language.

The fourth student, Mike, used a deterministic approach. He decomposed the language into three sublanguages. For the sublanguage corresponding to condition 1, he designed a correct DFA. He misinterpreted the two sublanguages corresponding to condition 2 and thus constructed two NCDFAs, accepting the languages $\{ba\}$ and $\{ab\}$, respectively (under this misinterpretation, nondeterminism indeed would not be helpful). He combined the two NCDFAs into an automaton for the concatenation language, simply by combining the final state of one with the initial state of the other. Next, he combined the DFA for condition 1 with the NCDFA of the concatenation language into an automaton for the union language. Mike did not use any known construction algorithm to combine these two automata. Instead, he designed a deterministic automaton, by combining the initial and final states and adding transitions to create some dependency between the automata. This process caused a logical error, due to the combination of the two final states into one. The resulting automaton is given in Figure 24.

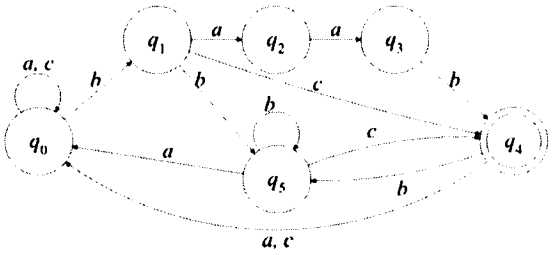

Figure 24. Mike's first full design

Mike was asked if he could think of any other automaton accepting the first sublanguage. He could not think of any such automaton, but when asked if he knew what a nondeterministic automaton was, he said he always preferred the deterministic model because it suited him. He described himself as a person with a tendency toward the exact sciences (physics) and in his opinion, nondeterministic thinking was not consistent with that.

When asked explicitly to design a nondeterministic automaton for the first sublanguage, he designed a noncomplete deterministic automaton (Figure 25), but immediately changed it by deleting the self loop with b in q_1 and adding b to the self loop in the initial state. The resulting automaton (Figure 26) is a nondeterministic automaton with two redundant transitions, thus matching the second pattern.

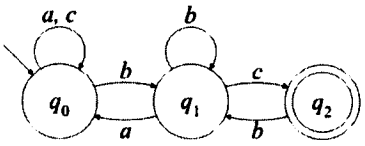

Figure 25. Mike's second design, part 1, version 1

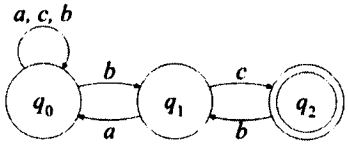

Figure 26. Mike's second design, part 1, version 2

Mike explained the last change by referring to the nondeterministic nature of this automaton, saying that it could read all the *b*'s in the initial state and move to q_1 only when the right *b* came along. When asked "Then why do you need these two transitions [the back-transitions with *b* from the final state and with *a* from q_1]?" he immediately realized that these transitions were redundant, deleted them and ended up with a fully nondeterministic correct automaton. Still, he said that for him deterministic automata were simpler.

Referring to the two NCDFAs that he designed for the other two sublanguages, we asked if the missing transitions did not pose a problem for him. He said that he had no problem with noncomplete deterministic automata since he knew he could add a sink state and direct all the missing transitions into the sink. That is, for Mike, the equivalence between the deterministic model and the noncomplete deterministic model is clear. Though the CM syllabus also includes the proof of the equivalence of the deterministic and nondeterministic models (using the subset construction), this is a more complicated and less intuitive proof. Even though Mike knew that these models were equivalent, the gap between them seemed too wide to make him comfortable with the nondeterministic model. Indeed, even when confronted with his misinterpretation of the other two sublanguages and asked to change his designs accordingly, he designed two correct but fully deterministic automata.

The next phase was to combine the two automata into one new automaton, accepting the concatenation language (corresponding to condition 2). Using the standard algorithm would have introduced nondeterminism. Mike did not use this algorithm, and combined them directly, resulting in a fully deterministic automaton.

Now Mike began to combine the last automaton with the nondeterministic automaton designed for the first sublanguage. He did so by changing his first version of this automaton, shown in Figure 24. The new version is shown in Figure 27. Again, Mike did not use the standard algorithm, which combined two NFAs into an NFA accepting the union language, but combined the automata by joining their initial states and their final states (this time, without adding transitions which create dependency), resulting, again, in an incorrect automaton (for the same reason as in the first version). During the design process he said to himself, "Yes, this one should be nondeterministic. It's simpler."

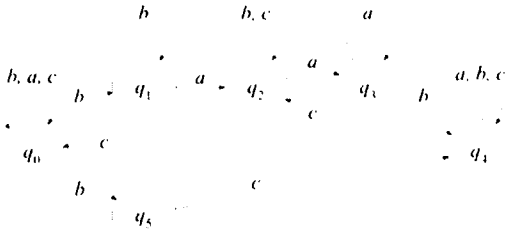

Figure 27. Mike's second full design, version 1

To confront Mike with the logical error in this automaton, we asked whether it was correct to add a self loop in the combined final state if such a loop does not exist in the final state of the first automaton. At first, Mike said that the nondeterministic nature of the automaton permits it. He said, "You can stay in q_0 until the string *bc* which ends the word comes along." This indicates a problem in understanding the asymmetric acceptance mechanism of NFA: One accepting path is enough to accept an input word, whereas all paths need to be rejecting paths in order to reject an input word.

Mike was asked to consider the input word *bca*. He immediately saw the problem and corrected the automaton by splitting the final state into two final states (Figure 28).

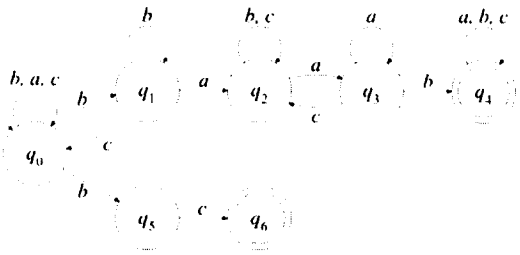

Figure 28. Mike's second full design, version 2

Note that when Mike designed the union automata, he said he was designing a nondeterministic automaton. The resulting automaton was not fully nondeterministic, and had many redundant transitions. In fact, Mike considered it nondeterministic since during the combination process he did not bother to add connecting transitions and left the two automata disjoint and independent except for joining their initial and final states. When asked why he chose to make it nondeterministic, even though he did not like nondeterminism, he said, "Otherwise, it would have been a mess, even more than it already is now." Indeed, Mike tried hard to keep his designs clean, with different colors for states, transitions, and input letters. He turned to the nondeterministic approach only when he felt that the design automaton might be too complicated, with higher chances for erasing things and creating "a mess."

At that point, Mike was asked to look again at one of the correct deterministic base automata he had designed for the sublanguages of condition 2, and to make it nondeterministic. He did that perfectly. Then he was asked to consider again the union automaton he designed (Figure 28) and make it "even more nondeterministic." Again, Mike did that perfectly, resulting in a fully nondeterministic correct automaton.

Mike's answers indicate a predetermined preference for the deterministic model over the nondeterministic one, irrespective of language. His answers imply that he sees the nondeterministic model as inaccurate, but he has no problem with the noncomplete deterministic model. Also, at least one of his designs indicates only partial understanding of the nondeterminism. Yet Mike appeared to be an intelligent student who understood his mistakes as soon as he was confronted with them, and corrected them quite easily. A preference such as his could perhaps be changed if the teacher emphasized nondeterminism and gave her students enough opportunities to practice it and gain a better perception of it. In this specific case, the teacher reported that she herself felt more comfortable with the deterministic model.

The answers given by the two students who chose to use the nondeterministic model showed that technical knowledge of the model does not necessarily reflect full understanding, or, in particular, a full perception of its computational capabilities.

Nondeterministic Pushdown Automata

The PDA model, introduced in Chapter 5 of the CM unit, is also nondeterministic. However, in the context of the CM unit, PDAs are usually more complicated than finite automata—they usually have more states and transitions. It is reasonable to assume that if students have difficulties understanding the nondeterministic mechanism when it is used with finite automata, they will have greater difficulty using the nondeterministic mechanism with PDAs. This may have two effects:

1. Students who choose to cling to the deterministic model, due to only partial understanding of the nondeterministic model, may be reasonably successful in designing DFAs for regular languages (even quite complicated languages) but may be less successful when designing deterministic PDAs for context-free languages. Again, since the PDAs presented in the CM unit usually have more states and transitions than DFAs, the chances to err when designing them is higher. Decreasing the dimensions of the automata by using nondeterminism could ease the design process and decrease the chance of error.
2. With only partial perception of the nondeterministic mechanism, students may choose a nondeterministic approach, but while they may be reasonably successful in regular languages, this may not be the case for context-free languages which induce more complicated automata. The resulting (nondeterministic) PDAs may contain errors.

Case Study

The following event, which took place in one of the classes, nicely demonstrates the second effect. The teacher had finished teaching the NFA model and was under the impression that the students had no problem with the perception of this model, and specifically with its acceptance mechanism. She began teaching the PDA model, and after defining it, she presented a PDA accepting the language $\{a^n b^n | n \geq 0\}$. This is a noncomplete but deter-

ministic PDA. The nondeterministic mechanism does not help in accepting this language. The PDA marks the first letter inserted at the bottom of the stack. This is a common technique, often used in the CM unit, and is essential for accepting certain languages, since, as mentioned earlier, the CM unit defined acceptance in the PDA model by reaching a final state (and not by emptying the stack). One of the students claimed that she could design a nondeterministic PDA, which doesn't need to mark the first letter inserted into the stack. She presented the following PDA (Figure 29):

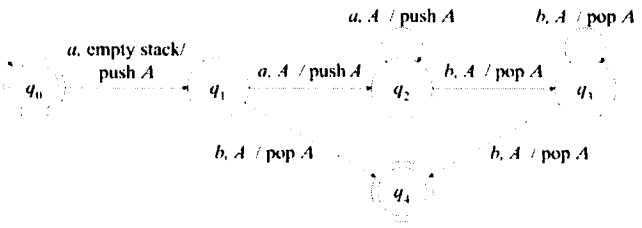

Figure 29. Student PDA

This PDA is indeed nondeterministic and it does not mark the first letter inserted into the stack. However, the problem with this automaton is that it also accepts words, which do not belong to the given language. The student demonstrated an accepting path for each word in the given language. All the other students agreed that this was a correct PDA for the given language, even after the teacher urged them to recheck it and look for a basic logical flaw. Not only did the students not notice that this automaton accepts words that do not belong to the given language, but even when the teacher confronted them with an accepting path for such a word, some of them said, "If a nondeterministic automaton can choose, then it can also choose a nonaccepting path for a word which doesn't belong to the given language."

These students' problem had nothing to do with the stack mechanism. It stemmed from an only partial understanding of the nondeterministic mechanism, specifically, assuming symmetry of acceptance and rejection of words. This partial understanding should have been discovered earlier, during the teaching process of the NFA model, but in this class it was discovered only in the context of PDAs, when the automata are bigger, and the number of possible paths increases.

SUMMARY, CONCLUSIONS, AND FURTHER RESEARCH

This article focused on the perception of nondeterminism, an important recurring concept in computer science and mathematics, relevant, for example, to probability theory, distributed computing, formal languages theory, and computability theory. The concept of nondeterminism is introduced in the CM unit integrated in a High-School curriculum.

The didactic considerations which guided the development of the CM unit were presented in this article: (a) Focusing on automata-based models (finite automata, pushdown automata, Turing machines); (b) Emphasizing theoretical aspects rather than implications in "real-life" fields (compilers, natural languages processing, etc.); (c) Introducing a new model—noncomplete deterministic automata (NCDFA)—which distinguishes incompleteness from nondeterminism, thus isolating the abstract concept of nondeterminism; (d) Omitting ε-transitions in all nondeterministic models; (e) Not integrating any simulation or visualization tools into the teaching process of the CM unit. The third and fourth considerations may guide the introduction of nondeterminism through computational models in the CS curriculum, the fourth being more suitable for the high school level, for which the students' ability of abstraction is less mature than that of university or college students.

In spite of the restrictions that guidelines a and d impose on the CM syllabus, the unit still succeeds in introducing meaningful theoretical discussions, which enable students to become acquainted with the rich theoretical foundations of computer science and the thinking patterns that characterize them. Specifically, the unit discusses the limits of computational power, nondeterminism and its contribution to computational power, comparison of various models in relation to computational power and closure properties.

Our findings regarding the use of nondeterminism show that a significant number of students (about a half for the first exams and about two thirds for the final exams) preferred to use a deterministic approach when designing finite automata. We believe that the level of use of nondeterminism reflects the level of students' perception of the concept of nondeterminism, since the teaching process of the CM unit explicitly emphasizes the advantages of the nondeterministic model. Our findings therefore imply that many students have not completely understood the concept of nondeterminism. The qualitative analysis of the students' solutions to the question on the first exam strengthens this conclusion. We saw that many students who used nondeterministic models when designing finite automata did not fully utilize them, and their designed automata also included deterministic characteris-

tics. In some cases, clinging to the deterministic model caused errors, which the students did not recognize, sometimes due to only partial understanding of the nondeterministic mechanism.

In addition, we found some evidence of differences between classes in the tendency to use nondeterminism. When interviewed, some of the teachers (even some who are considered leading CS teachers) said that they felt more comfortable with the deterministic approach, and therefore they did not emphasize the nondeterministic model and its advantages, beyond the minimal requirements of the syllabus.

We would like teachers to be aware of students' possible difficulties, and plan their teaching accordingly. A proper teaching process should strongly emphasize nondeterminism. Teachers should use additional examples and exercises to improve their students' understanding of nondeterminism. Naturally, the teachers themselves should use nondeterminism freely and consistently, both when first teaching the nondeterministic model and thereafter. This will probably help to demonstrate the advantages of nondeterminism, increase the level of exposure to nondeterminism and thus probably improve student comprehension. While teaching, teachers should try to asses each student's understanding of nondeterminism. To accomplish this, they can use the patterns introduced in this article. If a teacher sees a solution that matches one of these patterns, it may indicate a problem in the perception of nondeterminism, even when the solution is correct. In such a case, the teacher can help the student by giving him or her more focused examples and further focused practice. We also recommend that teachers introduce similar solutions, corresponding to the four patterns, in class, and highlight the differences between the solutions in relation to the level of nondeterminism.

The unexpected answers given by two of the students interviewed—that they didn't think that constructing a DFA was possible—indicate a problem in understanding the theory underlying the nondeterministic model. Even if students construct nondeterministic automata freely and correctly, the teacher cannot assume that they fully understand the theoretical meaning of this model. Specifically, students may not realize that the deterministic and nondeterministic models are equivalent. The teaching process should emphasize the theoretical aspects and not only the technical aspects.

To achieve these goals, this issue should be continuously addressed when preparing teachers to teach such a unit (both preservice and inservice training). Special effort should be made to ensure that teachers are appropriately exposed to nondeterminism and practice it thoroughly. We recommend introducing the four patterns and analyzing various examples corresponding

to them in preparatory courses. It is important to convey to teachers who intend to teach such a unit that the teaching process in class needs to properly emphasize the nondeterministic model in both its theoretical and its technical aspects, as previously described.

The effects of implementing such teaching strategies on students' perception of nondeterminism should be further studied, using classroom observations and interviews with teachers. Specifically, the role of teachers in the teaching process of this concept and the perception of nondeterminism among teachers should be investigated.

We recommend future research in additional various directions:

- The reasons underlying students' difficulties when learning nondeterminism: Mike said that he viewed nondeterministic models as inaccurate. This may be one of the reasons for the tendency to determinism. Another possible explanation relates these difficulties to the abstract nature of nondeterminism. Several studies in mathematics and computer science education (Hazzan, 1999, 2003a, 2003b) reported a tendency of students to reduce the level of abstraction, that is, to work on a level which is lower, more concrete, than the actual, appropriate level. This tendency can be easily followed when dealing with deterministic models, for which it is relatively easy to see a possible concrete implementation. But such a reduction in the level of abstraction is more difficult for nondeterministic models, which cannot be implemented as actual, concrete machines. Adequate research should validate these explanations, and search for other possible explanations.
- The effect of our didactic strategy: For example, does introducing NCDFA indeed have a positive effect on the perception of nondeterminism? The same question can be asked regarding the magic coin metaphor.
- The effect of the perception of nondeterminism in finite automata on the learning process of PDAs: We assume that in some cases, partial misunderstandings might not be discovered until the introduction of PDA. At that stage, relevant misconceptions which stem from this partial perception of nondeterminism, might be mistakenly assumed by teachers to reflect a partial misunderstanding of the stack mechanism.

Even though our results show that the concept of nondeterminism is a difficult one for students to understand, we believe it is essential for students to understand it properly since it is one of the basic topics of the CM unit, and an important recurring concept in computer science in general. Full understanding of the nondeterministic model can affect students' comprehen-

sion of other topics, such as pushdown automata and context free languages, threads and distributed computation.

As mentioned, nondeterminism is a basic computational and mathematical concept, yet, for many students, the CM unit we developed is currently the only part of their high school program which introduces the concept. The perception of nondeterminism has never before been examined in a high school or college and university context. We conducted this research with high school students, but since the issues are relevant to college and university students as well, we plan to conduct such a study on university students in the near future.

References

Armoni, M., & Gal-Ezer, J. (2003). Nondeterminism in CS high-school curricula. *Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference (FIE03)*, F2C (pp. 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)*, (pp. 17-21).

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

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

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.

Harel, D. (1987). *Algorithmics: The spirit of computing*. Reading, MA: Addison-Wesley.

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

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

Hazzan, O. (2003b). Reducing abstraction when learning computability theory, *Journal of Computers in Mathematics and Science Teaching*, (2), 95-117.

Hopcroft, J.E., & Ullman, J.D. (1979). *Introduction to automata theory, Languages and computations*. Reading, MA: Addison-Wesley.

Merritt, S.M., Bruen, C.J., East, J.P., Grantham, D., Rice, C., Proulx, V.K. et al. (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).

Tucker, A., Deek, F., Jones, J., McCowan, D., Sthepenson, C., Verno, A. (2003). A model curriculum for K-12 computer science. *Final report of the ACM K-12 task force curriculum committee.*

Acknowledgements

The authors would like to thank David Harel and Noa Lewenstein, consultants for the CM unit development team, for inspiring many of the didactic considerations described in this article.