A High-School Program in Computer Science

Judith Gal-Ezer[*] Catriel Beeri[†] David Harel[‡]
Amiram Yehudai[§]

Abstract

The authors are part of a committee that has been designing a new high-school curriculum in computer science and supervising the preparation of a comprehensive study program based on it. The new program emphasizes the foundations of algorithmic thinking, and teaches programming as a way to get the computer to carry out an algorithm. The paper discusses the program and its initial implementation.

Keywords: computer science, curriculum, education, high-school.

1 Introduction

Computers are without doubt the most important invention of the 20th century, having dramatically and irrevocably changed the way we live, and mostly for the better. One of the universally agreed upon implications of this is that educated people should be computer-literate. This, in turn, creates the need for introducing computers into high-school curricula.

[*]The Open University of Israel. (Part of this work was carried out during a sabbatical year spent at the Science Teaching Dept., The Weizmann Institute of Science, Rehovot, Israel.)

[†]Dept. of Computer Science, The Hebrew University, Jerusalem, Israel.

[‡]Dept. of Applied Mathematics and Computer Science The Weizmann Institute of Science, Rehovot, Israel. (Part of this author's work was carried out during a sabbatical at Cornell University, Ithaca, NY, and was partially supported by grants AF #F49620-94-1-0198 (to F. Schneider), NSF #CCR-9223183 (to B. Bloom), and NSF #CDA-9024600 (to K. Birman).)

[§]Dept. of Computer Science, Tel-Aviv University, Ramat-Aviv, Israel

However, computing as a scientific discipline, now called computer science (CS), predates the invention of computers. The first decades of the twentieth century saw a crystalization of fundamental concepts in this discipline, accompanied by discoveries on the limitations of computation that surprised mathematicians and philosophers. More recently, fueled in part by the invention of computers and their widespread use, the study of computing has bloomed, and CS is now recognized as an autonomous scientific discipline. Its scope includes the study and analysis of algorithmic processes, their power and limitations (sometimes called *algorithmics* [K, H]), as well as the design and implementation of computing systems. Its concepts are influencing work in other scientific disciplines, and, as in other scientific revolutions, its central notions and thought styles are becoming widely assimilated. Clearly, a modern high-school curriculum should reflect this growing importance.

Most high-school activity in computing has centered around courses in computer-literacy or the use of computers as a teaching aid in other disciplines. When computing has been taught in high-schoolas an autonomous subject, the emphasis has been most often on the technicalities of a programming language; at best, students learned to "code". However, coding is only one side of the coin. Our interest is far more foundational, and addresses the need to establish CS as an accepted scientific subject on the high-school level of education, to be taught on a par with other sciences, such as biology, physics and chemistry.

As in any scientific subject, one of the criteria for what constitutes the core of the subject is longevity. To a large extent, computer technology changes far more rapidly than the basic ideas of the *science* of computing, which center on the notion of an algorithm and its use in computing systems. This scientific facet, at least from the experience of the 60 years or so since the pioneering days of CS, has lasting and fundamental value. Thus, even though a proposed high-school program should enhance a student's ability to exploit computers beneficially, its backbone must be based on the science. Much of the insight, knowledge and skills it provides should be of value that is independent of specific computers and programming languages. Also, since the high-school years are the only period in which many students will be exposed to CS, a good curriculum must also aim at breadth and versatility.

This paper describes a new high-school curriculum in computer science, that has been proposed by a committee formed in 1990 by the Israel Ministry of Education. Its emphasis is on the basics of algorithmics, and it teaches programming as a way to get a computer to carry out an algorithm.

Section 2 provides some background for our work, Section 3 presents the curriculum itself, with its goals and principles spelled out in some detail, and Section 4 discusses our experience in implementing the ideas. Section 5 summarizes some topics for further work.

2 Background

CS is a relatively young discipline, and CS education is even younger. But it is not only the tender age of our field that causes problems for the educator. The nature of the field itself is a factor too. On the one hand, CS resembles mathematics, with its formal methods and abstract thinking, but on the other hand it is very much an engineering discipline, requiring concrete, down-to-earth skills. There have always been lengthy, often tedious, controversies on such issues as: What really *is* CS? What is its relationship with other fields? How do its subfields relate to each other? What characterizes a well-educated computer scientist? Some relevant references on these questions are [D, K, P, B].

There has been considerable activity surrounding CS curricula on all levels, and we have mentioned some of this work in a sidebar. Particularly notable is the high-school curriculum designed by ACM's special task force [M+]. Our work was independent of the ACM effort, and despite a similar basic philosophy, the goals and scope of the two programs, and the extent of the committees' mandate, are quite different: ACM's program amounts to a one-year, 120-hour CS-orientation course, whereas ours can reach 450 hours, taught over three years, and is really an extensive high-school-level study of the subject. Moreover, having been appointed by the body directly responsible for educational policy and its implementation, we are also heavily involved in supervising the many additional activities required to turn a skeleton curriculum into a widely-used working program. This includes the preparation of detailed courseware, the design of teachers' training courses, the delicate task of following an initial field test, and so on.

Our program was designed to fit Israel's particular educational needs, but appears to be applicable more widely. To help get a feeling for the context of our work, we have provided a brief description of the Israeli educational system in a sidebar.

Computer science has existed as a subject in Israel's high-school curriculum ever since the mid-1970s, but it has yet to become a fully accepted scientific subject like physics, biology or chemistry. Instead of the usual 3-unit and 5-unit programs studied towards the matriculation exams (see the sidebar), CS was taught in 2-unit and 4-unit programs. Moreover, many high-schools simply didn't offer CS at all, or offered the 2-unit program only. Partly as a result of this, the universities in Israel have never taken CS in high-school as seriously as the other sciences. In the overall evaluation of a university candidate, high-school units in CS are not worth as much as units in the other sciences.

The curriculum developed by a Ministry of Education committee in the 1970s was based on a solid and detailed course focused on programming in BASIC. In addition, it called for a rather extensive set of elective portions. If this curriculum were to have been implemented in full, it could have alleviated the problem somewhat, and by now we might have only had to update it to reflect changes in the field and a better understanding of the relevant educational issues. However, courseware (study books, teacher's guides, etc.) were not always available for the electives in the 1970s program. So that besides learning to program in BASIC, the students were not always exposed to the entire planned curriculum That committee did have algorithms in mind, but the emphasis in practice was on teaching and exercising the use of a programming language. In addition, some of the teachers themselves were less than adequately knowledgeable about the subject matter, and often taught the material as they saw fit.

Indeed, the "teacher issue", as we call it, is a very problematic one, and is visible from the start. While there are well-defined requirements for qualifying teachers in most other high-school subjects, the situation in CS is quite different. No self-respecting school system will leave its physics or mathematics program in the hands of a self-taught instructor, or one with only high-school education. Yet this is exactly what very often happens in CS programs in many countries. It was only in 1992 that the USA organization NCATE (National Council for Accreditation for Teacher Education) adopted standards for teacher preparation programs, to take effect starting in 1994 [TTK].

Getting back to the development of curricula in Israel, various new units based on Logo and an electronic spreadsheet language were developed in the 1980s (among other things to meet the growing demand for teaching computer literacy). Some schools adopted these in place of parts or all of the curriculum, which sometimes entailed moving even further away from true computer science.

Our committee was formed in 1990. It consists of a researcher specializing in the educational aspects of mathematics and computer science (the first-listed author of this paper), three computer scientists who are involved in educational issues on various levels (the remaining co-authors of this paper), two experienced high-school teachers of computer science, three education professionals from the Ministry of Education, two of them from the computer and computer science section, and the head of that section himself.

We started out by reviewing the existing situation, and concluded that the whole issue must be re-addressed, and a new and carefully thought-out computer science program must be developed for high-school (i.e., grades 10 through 12). We were convinced that the committee should not only decide on the general topics and principles, but should also prepare detailed and rigorous syllabi for all units in the program; it should help form and supervise the teams who prepare the courseware, providing them with continuous technical feedback; it should be involved in the development of teacher training activities; and it should guide and follow a small-scale initial implementation of its recommendations.

3 The New Program

3.1 Underlying principles

Before getting into a more detailed description of the new program, it is helpful to pinpoint the principles we have used to guide our work, some of which recapitulate issues discussed above. In reading them, it helps to keep in mind that the program must introduce a new subject; students are not exposed to any computer science before embarking on it. This is one of the main differences between computer science and other high-school subjects.

- *Computer science is a full-fledged scientific subject.* It should be taught in high-school on a par with other scientific subjects.
- *The program should concentrate on the key concepts and foundations of the field.* The main notion to be emphasized throughout the pro-

gram is that of an algorithmic problem, and an algorithm as a solution thereof. To some extent, the more general notion of a system, and the accompanying principles of modularization and abstraction should also be discussed. Other topics are to be viewed as building upon these.

- *Two different programs are needed, one for 3 units and one for 5.* The first is for students with only a general interest in CS, and the second, which should be deeper and broader, is for those with more specific interest in CS. However, the design of the 3-unit program should take into account that for many students this might be the only exposure to computer science, so that some attempt at comprehensive coverage should be made.

- *Each of the two programs should have required units and electives.* While the entire program should consist of central and important topics, some of these are less crucial than others and can be made elective. Moreover, variance and flexibility is important for its own sake.

- *Conceptual and experimental issues should be interwoven throughout the program.* The word 'conceptual' here does not mean 'impractical'. It refers to subjects that are taught in the classroom, rather than in the laboratory. This two track approach, which we dubbed the "zipper principle", is one of the salient points of our program, and is elaborated on in a sidebar.

- *Two quite different programming paradigms should ideally be taught.* It is highly recommended that a student should learn a "mother tongue" first, but then, on a more humble scale, be introduced to another language, of radically different nature, that suggests alternative ways of algorithmic thinking. This emphasizes the fact that algorithmics is the central subject of study.

- *A well-equipped and well-maintained computer laboratory is mandatory.* This is the responsibility of the school system, and entails setting things up to support laboratory sessions and adequate individual "screen-time" for students.

- *New course material must be written for all parts of the program.* The teams that are to prepare the courseware must have "real" computer scientists on board, as well as CS high-school teachers and researchers in computer science education.

- *Teachers certified to teach the subject must have adequate formal CS education.* An undergraduate degree in computer science is a mandatory requirement, as is formal teacher training.

The program should focus on the most basic and lasting concepts of CS. It must be challenging, in the sense that it will not only teach the foundations, but will also relate them to the practical side of computing, and it should train the students to deal with intellectually demanding tasks.

3.2 Structure and Contents

The program comes in two versions, a 3-unit one and a 5-unit one. The former consists of 270 hours of study, and the latter of 450. (All hours in the paper are absolute. The way they are spread out over days and weeks is determined by the schools.) The programs are constructed, as explained below, from the following list of modules:

1. **Fundamentals 1** and **2** (2 units; 180 hours): This module pair provides the foundation for the entire program. It is the one that introduces most of the aforementioned central concepts, and in a parallel realization track teaches how to apply them in a procedural programming language.

2. **Advanced programming** (1 unit; 90 hours): This module is really a continuation of *Fundamentals.* It concentrates on data structures, introducing abstract data-types in the process, and also takes a step beyond stand-alone algorithms, to discuss the design of complete systems.

3. **Second paradigm** (1 unit; 90 hours): This module introduces the student to a second programming paradigm, which is conceptually quite different from the procedural approach adopted in modules 1 and 2. Currently approved are logic programming and system-level programming units; other possibilities include object-oriented programming, functional programming, and concurrent programming.

4. **Applications** (1 unit; 90 hours): This module concentrates on one particular kind of application, teaching both principles and practice. Currently approved alternatives are computer graphics and management information systems.

5. **Theory** (1 unit; 90 hours): This module is intended to expose the student to selected topics in theoretical computer science. Currently approved are two alternatives: A full unit on models of computation (mainly automata), and a two-part unit consisting of models of computation and numerical analysis.

The two *Fundamentals* units are mandatory for both programs. For the third unit in the 3-unit program there is a choice between the *Second paradigm* and *Applications*. In the 5-unit program, *Advanced programming* is mandatory, and the fourth and fifth units are chosen from among the *Second paradigm*, *Theory* and *Applications*. Since *Second paradigm* and *Applications* can be taken on the 3-unit level too, we envision somewhat deeper versions of them being developed for the 5-unit program in the future.

3.3 The modules

We now describe the contents of the modules:

Fundamentals 1

This is the basis of the entire program, and is taught in the 10th grade. It is also intended to be usable as a stand alone mini-course for students who will not study any more computer science. It covers the basic concepts of an algorithmic problem and its solution — the algorithm. It also discusses functions as a refinement mechanism, and introduces the notions of algorithmic correctness and efficiency. The contents of this module is described in more detail in a special sidebar.

The sidebar on the zipper principle explains the underlying pedagogical approach taken in this and other modules. Basically, each subject is introduced first on a conceptual level, including manual exercising, and is then recast in practical, implementational form using a programming language.

Much has been said about the significance of the first programming language — the "mother tongue"; see e.g., [W, Ba, LP]. Many good arguments have been made for adopting non-procedural styles of programming in the first course, notably functional languages such as Scheme [AS]. Since this is still controversial, and there is no clear agreed-upon approach emerging, we became convinced that for high-school it is probably best to remain in the mainstream, and adopt a procedural (imperative) style of programming. We thus decided to adopt a "vanilla" procedural approach, and, as of now, we support development of a Pascal version of the material. However, the program itself does not impose a specific language, and the future might find teams developing courseware that uses other languages. Since much of the curriculum is actually language independent, we shall not dwell on the language issue any further.

Fundamentals 2

This second part of the basic material is taught in the 11th grade. It first revisits and expands upon some of the topics covered in the 10th grade, in order to deepen the student's understanding of that material. A number of new facets of algorithmic analysis and design are emphasized, such as stepwise refinement, top-down and bottom-up techniques. In addition, the following topics are taught: Recursion (only for 5-unit-ers), procedures, and two-dimensional arrays. Time efficiency is treated in more detail, and a special section is devoted to more advanced problems, such as searching and sorting.

Advanced programming

This is a continuation of *Fundamentals*. In universities, the course that follows introductory computer science is usually devoted to data structures and data types. We had more general goals in mind, namely to endow the student with a basic understanding of larger systems and their organization principles. New data structures are taught, such as stacks and binary trees. Whereas in the *Fundamentals* module procedures and functions were the main structuring tool, here abstract data types are added to help in handling larger systems. Dynamic memory allocation is touched upon, and the module also involves the implementation of one or two small systems.

Second paradigm

Currently, there is a logic programming module available, and another module based on assembly language is in preparation. The first of these introduces basic notions from logic, and discusses knowledge representation by facts and clausal rules. Programming is carried out in Prolog, with recursion, lists and trees taking a prominent place in the material. The second presents the conceptual structure of a computer system, and provides an introduction to programming in assembly language. We would very much like to see additional alternative modules developed for this unit, such as ones based on functional or object-oriented programming, or a unit developed around concurrency.

Applications

In deciding upon the contents of this unit, we were motivated by the need to cater to students in the special technological track.[1] The unit should help them in applying computers to the profession they are pursuing. The most relevant non-CS specializations in this track are those that revolve around information systems (for example, in hotel administration), or graphical aspects of computation (for example, in architectural design). Accordingly, two modules are being developed. One is an introduction to management information systems, which discusses logical file and data organization, a system's life cycle, and basic systems modeling and analysis. The other is an introduction to computer graphics, which deals with the basics of representing and manipulating graphic objects, and their use in problem solving. In both modules the student gets to use a ready-made software package, such as a database system or a CAD package, and a final project is required. Here too, we would like to see additional alternative modules developed in the future.

Theory

This unit exposes the student to topics in theoretical computer science. Two modules are currently under development, one in models of computation and the other in numerical analysis. The first introduces finite automata, pushdown automata and Turing machines, and elaborates on their relative power. It also presents the Church-Turing thesis, and briefly discusses the limitations of computers. The second module concentrates on two main topics: Iterations for root extraction, and the solution of linear systems of equations. Issues treated include round-off errors, absolute and relative errors, approximate solutions with error control, and ill-conditioned problems. Two versions of models of computation are being developed, one of 90 hours,

[1]However, the unit can be taken by all students.

which covers the full unit, and the other an abridged version of 45 hours, which is taken together with the 45-hour numerical analysis module.

4 Getting the Program Under Way

4.1 Developing the material

When we had the first version of the curriculum planned out, we proceeded by appointing professional teams to prepare detailed syllabi, to be followed, after the committee's approval, by the development of course material.[2]

A typical development team comes from a science teaching department of an Israeli university, and has three to four members. We insisted that there be at least one computer scientist on board, one academic researcher with experience in CS education, and one high-school teacher of CS. We tried to distribute the choice of teams, so that as many academic institutions as possible would take part in the effort. This helps ensure that the program is versatile, in the sense that different scientific and didactic approaches are represented.

Preparation of the syllabi for the various modules was quite a lengthy process, despite the fact that many of the topics are taught at the university level, where we could draw upon accumulated experience. Indeed, several versions of the syllabi were often needed before the committee was able to give its final approval. In addition, the syllabi were often further changed during the period of courseware preparation, and even during the field test. Still, the preparation of syllabi turned out to be straightforward in comparison with the preparation of the courseware itself.

One of the first difficulties we faced in developing the program, which became more acute in writing the courseware, was that of student population. Ideally, we would have preferred to develop separate courseware for the 3-unit and 5-unit programs, reflecting the significant difference in required breadth and depth. In reality, the way the schools are set up in Israel does not encourage this, as students are not required to make decisions on program alternatives in *any* subject until just before the 11th grade. In fact, in the 10th grade, when our program starts, many students have not yet decided

[2]The members of the committee were, and still are, deeply involved in all aspects of the preparation of syllabi and courseware, and also in some aspects of the implementation.

whether they will be taking computer science for matriculation at all. In the 10th grade, a typical science-oriented study group studies all the main scientific subjects available in high school, i.e., physics, chemistry, biology, and in some cases computer science. Students make their actual choice of matriculation subjects only towards the 11th grade. Hence, we had to plan things such that part of the material would be accessible to a heterogeneous collection of students, including future 5-unit-ers as well as ones who are not fit even for the 3-unit program.

Since computer science studies in the 10th grade usually involve three weekly hours, giving a total of 90 hours for the year, we had to make the first 90 hours of study, i.e., the *Fundamentals 1* module, not only sufficiently elementary, but also of stand-alone nature. By this we mean that those students who will end up choosing not to continue CS beyond the 10th grade, will have been given a well-rounded, if simple, view of the important aspects of the subject. The description of *Fundamentals 1* in the sidebar reflects this.

Another problematic aspect of the development of courseware was our own rather severe underestimation of the effort. On the one hand, as a collection of individuals, our committee has extensive experience both in high-school and university teaching and in writing CS textbooks and courseware. On the other hand, we should have known that projects like this always take longer, and are more painful and tedious, than one plans. Anyway, we decided to start the development of a core program, consisting of the mandatory modules but only a small number of the electives. Our rationale was that the availability of good core material would help the program be adopted by the Ministry of Education officially as *the* CS program in high-school, and put into operation in many schools. Then, we reasoned, there would be sufficient support to drive the development of additional modules as needed.

In retrospect, the decision to develop only a core program first was fully justified, among other reasons because our optimism was greatly exaggerated. Now, four years later, we still do not have final versions of course material for even the core program. Had we tried to shoot for a maximal program from the start, the situation might have been even worse. The good news is that at present we have satisfactory syllabi for most of the aforementioned modules, and courseware for several of them that is either satisfactory or is at least good enough to get us through the initial period. Improvements are still needed, of course, since we cannot claim to have final versions until the material is thoroughly class-tested, and also possibly rewritten.

Courseware preparation on such a large scale clearly requires significant budgets. The way things are set up in Israel, committees such as ours do not have operating budgets, and can only make recommendations to the Ministry of Education. This means that the success of the proposed program depends to a large extent on our continuous ability to convince the Ministry people that it is worth their investment.

4.2 Teaching the material

In the Fall of 1991 we started a very limited field test for parts of the new program. It first involved 8 study groups in 5 schools, and by 1994 it has grown to around 40 groups in 9 schools. These small numbers reflect the difficulty of convincing schools to participate in a new and embryonic program, but also that of finding good teachers.

As mentioned above, the teacher problem turns out to be one of the main difficulties in implementing the program. The anomaly is that most CS teachers do not have a university degree in computer science. This is unacceptable, and one of the committee's decisions has been to require such a degree from any teacher seeking a permit to teach CS in high-school .[3] Specifically, we require a B.Sc. or its equivalent in CS, or an appropriate B.Ed. in CS education.

While we are confident that, if followed, this policy will bring about a gradual improvement in the overall quality of CS teaching, it does not solve our present problems. There is a large group of teachers who have been teaching programming and CS in high-schools for years without such formal training. Some of these might quit when the new program becomes fully operational, but the majority will probably want to continue. For these latter ones, we have outlined a special crash course, consisting of around 6 basic subjects that are taught in CS departments in the universities.[4] Teachers without CS degrees will be required to complete these courses in order to be allowed to teach the new program.

[3]The Ministry of Education official regulations, rarely adhered to, are that a teacher is required to have a *second* degree (M.Sc. or its equivalent) to teach a full scientific program in the 11th and 12th grades of high-school . However, in view of the job market situation, we have tried to be more realistic.

[4]In Israel's Open University these subjects can be taken for credit without necessarily being part of a degree program.

In addition, even the best of teachers will need some training in order to teach the modules of the new program. For example, not every CS graduate is familiar with logic programming or information systems, and even those who are could do with help in the didactic aspects of teaching such topics in high-school. The few teachers chosen for the field test, although better trained than others, still had to spend considerable time and energy in ad-hoc teacher training, usually provided by the developers of the courseware in accordance with the committee's guidelines. This was necessary, in part, to dispel some beliefs and habits of the participating teachers: In the initial phases of the field test many of them felt more confident teaching what they knew well, like the technicalities of the programming language. Also, some teachers expressed disapproval of changes in emphasis. For example, we de-emphasized traditional flowcharts, and they felt they were being deprived of an important technical device; we emphasized pseudo-code in algorithm design (as opposed to direct coding), and they felt this new medium was too vague.

Although the ad-hoc training greatly helped the initial group of teachers, it seems certain that within a couple of years we will have to develop a massive instructional program for the general body of teachers who want to get involved in the new program. This is one of the most challenging problems the committee faces in the future, and will require lots of attention.

Besides the teacher problem, there are other difficulties that have surfaced in this initial implementation. One is the incredibly diverse backgrounds of the students. On the one hand, the program must cater to students with no familiarity with computers at all (except possibly for computer games). On the other hand, many students have extensive programming experience. What makes the problem particularly acute is that most of these are self-taught hackers, and have developed habits that are often a severe hindrance to the orderly study of algorithmics.

Some schools are short of computers and cannot provide students with the necessary laboratory time; the labs are also inadequately maintained. There is still a long way to go before schools treat their computer labs with the same respect they confer on their physics and chemistry labs.

Another objective difficulty is the fact that Israeli universities do not yet give incoming students the same bonus points for high-school computer science as they do for other sciences. Until this happens, many potential students will hesitate before agreeing to choose computer science, resulting in a troubling Catch-22 situation.

As to the field test itself, the jury is not in yet. Our current feeling, however, is that so far it has been quite successful. Judging from teachers' reports and the results of student exams, it seems that the main goals are being met. The schools themselves have initiated a three-fold increase in the number of classes joining the field test since its inception.

5 Postscript

In summary, we are quite happy with the way the program is developing, although our initial expectations were somewhat higher. We hope that within two years it will be adopted throughout the Israeli high-school system. Nevertheless, there is still lots of work to do. Some parts of the courseware are not ready yet, and we feel that even the parts that are will have to be rewritten in a few years, to reflect experience gained in wide-scale application. In addition, we mentioned the need to develop two versions of *Fundamentals*, and new alternatives for the *Second paradigm* and *Applications* modules. We must also devise effective teacher training courses.

Our work can be viewed as providing a basic no-frills program for computer science study in high-school. With this in mind, we envision more specialized variants being developed in the future, perhaps based on different didactic approaches, to deal with heterogeneous groups of students or with project-oriented study. Finally, we strongly believe that the program should evolve to use specialized educational software (e.g., as in [BE]), and to take advantage of state-of-the-art technology, such as interactive TV.

Acknowledgements: The work reported upon here reflects the efforts of many people. First and foremost are the members of the committee itself, which is chaired by Amiram Yehudai. Besides ourselves, it includes Ben-Zion Barta (who heads the computer and computer science section in the Ministry of Education), Roni Dayan, Ephraim Engel, Meir Komar, David Levant, and David Sela. We also owe a great debt to the members of the various development teams, too numerous to list individually, whose work and dedication permeate the entire effort.

References

[AS] Abelson, H., and G. J. Sussman, *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, MA, 1985.

[Ba] Baranauskas, M. C. C., "Observational Studies about Novices' Interaction in a Prolog Environment Based on Tools", *Proc. 7th Int. PEG Conf.*, Edinburgh, pp. 537–549, 1993.

[BE] Barwise, J. and J. Etchemendy, *Turing's World*, CSLI Publication, Stanford, CA, 1993.

[B] Biermann, A. W., "Computer Science for the Many", *Computer* **27**:2 (1994), 62–73.

[D] Dijkstra, E. W., "On the Cruelty of Really Teaching Computing Science", *Comm. Assoc. Comput. Mach.* **32** (1989), 1398–1414.

[H] Harel, D., *Algorithmics: The Spirit of Computing*, Addison-Wesley, 1987 (2nd ed., 1992).

[K] Knuth, D. E., "Computer Science and its Relation to Mathematics", *Amer. Math. Monthly* **81** (1974), 323–343.

[LP] Lee, A. and N. Pennington, "The Effects of Paradigm on Cognitive Activities in Design", *Int. J. Human-Computer Studies* **40** (1994), 577–601.

[M+] Merrit, S., et al., *ACM Model High School Computer Science Curriculum*, Association for Computing Machinery, New York, 1994.

[P] Parnas, D. L., “Education for Computer Professionals”, *Computer* **23**:1 (1990), 17–22.

[TTK] Taylor, H. G., L. G. Thomas and D. G. Kneze, “The Development and Validation of NCATE-Approved Standards for Computer Science Teacher Preparation Programs”, *J. Technology and Teacher Education* **1** (1993), 319–333.

[W] Wexelblat, R. I., “The Consequences of One’s First Programming Language”, *Software – Practice and Experience* **14** (1981), 733–740.

6 Sidebar – Some other CS programs

Over the past years, there has been a steady evolution of university-level curricula, starting with Curriculum ’68 [ACM], followed by subsequent versions, such as [T+]. On the high-school level things have been slower, due in part to the failure in distinguishing adequately between computer science, computing technology (i.e., issues concerning specific systems) and general computer literacy. Over the years, there have been efforts to establish high-school curricula in computer science, see e.g., [R+], as well as relevant teachers’ training courses [P+, MJH, TTK]. Notable in this is ACM’s task force, whose recommended curriculum has been published very recently [M+]. In the text, we touch upon the differences between this work and ours.

References

[ACM] ACM Curriculum Committee on Computer Science, “Curriculum ’68 Recommendations for Academic Programs in Computer Science”, *Comm. Assoc. Comput. Mach.* **11** (1968), 151–197.

[M+] Merrit, S., et al., *ACM Model High School Computer Science Curriculum*, Association for Computing Machinery, New York, 1994.

[MJH] Maddux, C. D., L. Johnson and S. Harlow, “The State of the Art in Computer Education: Issues for Discussion with Teachers-In-Training”, *J. Technology and Teacher Education* **1** (1993), 219–228.

[TTK] Taylor, H. G., L. G. Thomas and D. G. Kneze, "The Development and Validation of NCATE-Approved Standards for Computer Science Teacher Preparation Programs", *J. Technology and Teacher Education* **1** (1993), 319–333.

[T+] Tucker, A., et al., "Computing Curricula 1991: A Summary of the ACM/IEEE-CS Joint curriculum Task Force Report", *Comm. Assoc. Comput. Mach.* **34** (1991), 69–84.

[P+] Poirot, J., et al., "Proposed Curriculum for Programs Leading to Teacher Certification in Computer Science", *Comm. Assoc. Comput. Mach.* **28** (1985), 275–279.

[R+] Rogers, J., et al., "Computer Science for Secondary School: Course Content", *Comm. Assoc. Comput. Mach.* **28** (1985), 270–274.

7 Sidebar – Israel's education system

The Israeli education system is basically centralized. The Ministry of Education sets educational policy on all levels, and then implements it with the aid of specialized committees, work teams and professional supervisors.

Students go through nine years of mandatory education, usually divided into elementary school (grades 1–6), and mid-level school (grades 7–9). Three additional high-school years (grades 10–12) are optional. These three years culminate in an extensive set of matriculation exams (called "bagrut" in Hebrew), which, among other things, are crucial for for admission into Israeli universities. The exams are based on a core of required subjects, and several additional electives. Subjects are taught in "study units", each of which denotes the equivalent of three hours a week for a year, or approximately 90 hours. To get through the matriculation hurdle, a student has to successfully pass the exams in at least six subjects, accumulating a minimum total of 20 study units, though most students take more.

Many of the subjects can be studied on various levels, the most common of which follow 3-unit and 5-unit programs, that differ significantly in the quantity of the material and the conceptual depth of the presentation. A 5-unit program would typically require studying the subject for 5 weekly hours throughout the three years of high school. The required courses are Hebrew (language and literature), English, Bible studies, mathematics and history, and there are several electives, such as talmud and geography. The sciences are also elective, and include physics, biology and chemistry. In some schools there is also a technological track, in which students study some technical subject intensively, with the goal of preparing for specializing in it later on in life.

Obviously, no two educational systems are quite the same. Nevertheless, high-school studies in most countries contain a scientific component that is comparable with — although in some places not as extensive as — that of the Israeli system. It appears, therefore, that our work could be applicable to other countries too.

8 Sidebar – The zipper principle

A major principle in the design of our program, and a crucial guideline for teaching it, concerns the interweaving of conceptual and experimental issues. We call this the "zipper approach" — a little of this, followed by a little of that, and so on, combining to form a unified whole.

This "zippering" is most visible in the two *Fundamentals* modules, as well as in *Advanced programming*. In fact, these three modules, taken together, constitute a two-track effort, the one conceptual and the other a recasting of the concepts and ideas in practical, implementational form in a real programming language.

Progress along the two tracks is made in parallel. Each new concept is first discussed in the classroom. Then, if needed, relevant parts of the programming language are introduced, and the student gets to practice them in the computer laboratory. Manual problem solving is carried out prior to the implementational segment, a particularly important matter, as we want the student to understand that the concepts are more fundamental than their specific realization in a particular language. He or she will become more acutely aware of this later, when exposed to an additional programming paradigm, but we want to drive the idea home from the start.

For example, we recommend that the notion of repetition in algorithms be illustrated with informal, natural language descriptions (such as "capitalize all words in the input list") before any programming language renditions of it are introduced. A specific programming construct (for repetition it is the `while` or `for` statement) is then presented, and the students apply and practice their knowledge using it. Thus, we do not teach the `while` statement as an entity in its own right, but, rather, as one of many possible forms a repetitive construct can take on in a programming language. Even so, we do not spend *too* much time discussing the abstract notion, since most high school students need the concrete in order to fully understand the abstract.

The zipper principle is reflected in the final matriculation exam too, part of which takes place in the school's computer laboratory.

9 Sidebar - The *Fundamentals 1* module

The *Fundamentals 1* module is taught in the 10th grade. Besides constituting the basis of both the 3-unit and 5-unit programs, it is also intended as a stand-alone mini-course for those students who choose not to continue with computer science at all.

The following list of its main topics reflects the experiences of three years of experimentation (see Section 4), and might still undergo some minor changes.

We should remark that most of the topics in the module are not simply taught, exercised, and then set aside. They are listed here in linear order, and with their recommended hours, to show when and how extensively they are first introduced. In actuality, most of them are really "wall to wall" topics, and accompany the material of the entire module (as well as that of *Fundamentals 2* and *Advanced programming*), with varying intensity, all along.

- (5 hours): Introductory notions, such as algorithms, algorithmic problems, and the execution process.

- (15 hours): Introducing the basic computation model of data, variables and input/output. Emphasis is placed on viewing a simple program as a sequence of value-changing instructions.
- (3 hours): Modularization — constructing an algorithm from simpler ones.
- (9 hours): Conditional execution; Boolean conditions with ‘and’ and ‘or’ connectives.
- (4 hours): An initial discussion of the correctness of algorithms, mentioning valid inputs, correctness with respect to an algorithmic problem, and testing by running on sample inputs.
- (12–15 hours): Repetitive execution of various kinds; counters; accumulators; exit conditions; nontermination.
- (3 hours): An initial discussion of the time efficiency of algorithms, including running time as a function of the input, comparing running times, and worst-case time behavior.
- (8 hours): Functions, emphasizing the use of a function to solve a subproblem, and viewing a function call as a new basic instruction.
- (12 hours): One-dimensional arrays.
- (16–19 hours): A section that concludes the 10th grade material, and includes more complex examples of algorithms and programs, the nesting of control structures, and more.