FACILITATING HOW TO LEARN ALGORITHMS AND COMPUTER PROGRAMMING

Osvaldo Luiz de Oliveira[1]

Abstract — *Computing devices are increasingly present in everyday objects and activities, this makes computer programming an essential issue for modern Engineering. The importance and complexity of this issue has motivated the research on educational methods to facilitate the learning of this subject. However, current methods have, in general, the following drawbacks: (1) they expect a learner able to formulate strategies for solving problems even in fields not know properly; (2) the used languages have abstract primitives, distant from the concepts belonging to the areas where the solutions are formulated. In this paper, we present a method for learning algorithms that exposes the student to different problems spaces, each one with its own formal language. These problems spaces evolve gradually: from small to large spaces, from concrete to abstract spaces, from specific to generic language. We also present a software developed and some results in classroom of Engineering and Computer Science.*

Index Terms — Algorithms and Programming Learning, Computer Science Education, Engineering Education.

INTRODUCTION

There was a time when programming was used only by engineers to automate calculations and to support the development of projects. Nowadays, the computing devices are present in virtually every object and every human activity. The incorporation of computers by traditional artifacts of modern culture, as the automobile, television, and telephone, and the integration among computers and physical space allowing the construction of "smart buildings", for example, makes computer programming a matter of great importance for Modern Engineering.

However, computer programming is a notoriously difficult task. Researches show that the dropout rate for students in first and second year in Computer Science courses, due to difficulties in programming learning, is between 30% [10] and 70% [6]. The importance and complexity of this issue has motived the research of educational methods to facilitate the programming learning. The existing proposals emphasize partial aspects involved in the learning process of the development of algorithms, without consider the problem globally: syntactic aids [10]-[17], simplification of IDEs [2]-[7], visualization of programming concepts [8]-[18], emphasis on programming on domains of practical problems [4]-[5]-[10]-[11]-[13], didactic aspects encompassing curriculum [9], monitoring [9], assessment [9]-[18], teacher´s feedback [7], automatic evaluation of solutions [6]-[7], iconic notations for the execution flow [3], assignments [15]-[16]-[19], list of common misconceptions [14], collaborative learning [1]. However, the existing proposals do not address an appropriate model of how the learning can take place gradually from the most basic concepts to the most complex.

In this paper we propose a logic framework to explain the factors that affect the programming learning. It allows measuring the complexity of a particular knowledge domain and of a specific programming language. Based on this logic model, we developed different spaces of problems and languages through which students may develop the knowledge gradually. Finally, we show some experimental results obtained in the classrooms of Engineering and Computer Science and present our conclusions.

SPACE OF PROBLEMS, FORMAL PROGRAMMING LANGUAGE, IMMEDIATE AUTOMATON, ALGORITHM AND PROGRAMMING ENVIRONMENT

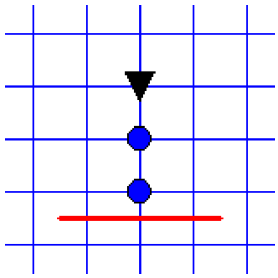
A **space of problems**, or **problems' domain**, comprises an area where problems can be proposed. A space of problems includes a theme and involves objects and concepts utilized to formulate problems. One **problem** describes a proposition, a question, a task to be solved by an **algorithm**. To describe the solution of a problem in a space of problems, the student must use a **formal programming language**. The **programming environment** includes, among others, a set of tools used to edit, compile, run, and debug the algorithm. As an example, an environment configuration for programming learning, widely used, has Mathematics as a space of problems, Pascal as a formal programming language and the IDE of Free Pascal as a programming environment. The configurations described below have a key role in our proposal and will be mentioned throughout this text.

World of Robots, Language from the World of Robots

The World of Robots is a space of problems that involves a rectangular board representing a "World", robots that can move around the "World" and some other objects: walls and discs. The World of Robots induces a space of problems linked to the movement and manipulation of objects, disks, on a board possibly consisting of barriers formed by walls (Figure 1-a). The Language from the World of Robots consists on traditional flow control structures and commands

1 Osvaldo Luiz de Oliveira, Faculty of Campo Limpo Paulista - Faccamp, Rua Guatemala, 167, Jardim América, 13231-230, Campo Limpo Paulista, SP, Brazil, osvaldo@faccamp.br.

such as "GoFoward (r)", "TurnLeft (r)", "CatchDisc (r)", "PutDisc (r)" which causes a robot named "r", respectively, to walk one step forward, turn 90° left and pick up or place a disc in the position where it is. Other commands allow you to check if a robot named "r" is on a disc, "IsOnDisc (r)", and if in front of the robot there is no wall, "FrontIsClear (r)". Figure 1-b describes an algorithm to solve the problem of collecting all discs in the front of the robot named "r" until he finds a wall. The programming environment consists on an editor that allows you to create and place robots, discs and walls and a text editor to enter the algorithm. In response to the request of the algorithm execution, the programming environment displays the animation of programmed movements.



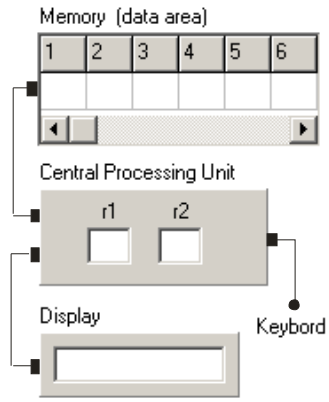
```
while FrontIsClear (r) do
begin
  GoForward (r);

  if IsOnDisc (r)then
    CatchDisc (r)
end
```

(a) (b)

FIGURE 1
LEARNING ENVIRONMENT "WORLD OF ROBOTS"

Integer and Boolean Arithmetic, Simple Computer

Assembly Language: addition, subtraction, multiplication, division and boolean operations form the basis of this space of problems. The formal language operates on the simple computer model, illustrated in Figure 2-a. For simplicity, a Simple Computer's memory has exclusive areas for data and for the program. The data area has 64 Kbytes, addressed in decimal numbers from 1 to 65536, again for simplicity. The processor has two registers, named "r1" and "r2" and is capable of receiving data from a keyboard and send data to a numeric display. The language grammar includes the instructions that may be performed by the Simple Computer's processor. Some of them are: "input" that receives an integer from the keyboard and stores it in register "r1"; "output" that sends the integer of the register "r1" to the numeric display; "move" that can copy data between the memory and registers; "add" that adds the integers stored in registers "r1" and "r2" and places the result in "r1". Figure 2-b presents an algorithm for the problem of adding two integers. The programming environment implements the editing, execution, and viewing the algorithm operation, instruction by instruction.




```
input
move r1, 1
input
move 1, r2
add
output
```

(a) (b)

FIGURE 2
LEARNING ENVIRONMENT "*INTEGER AND BOOLEAN ARITHMETIC, SIMPLE COMPUTER ASSEMBLY LANGUAGE*"

Integer and Boolean Arithmetic, Arithmetic Language

The space of problems here is the same as before. The language we are here referring to as Arithmetic Language is a subset of the traditional algebraic notation. So, "+", ".", "∑" and "*x*" represent, respectively, the operations of addition, multiplication, the sum of a set of terms and a variable named "*x*". On a form, algebraic expressions can be written by an editor specifically designed to facilitate the editing of algebraic symbols (Figure 3-a). Such as manual calculations carried out on a paper, a "virtual machine" calculates the expressions (runs the algorithm) from top to bottom and from left to right. The execution of the algorithm for $m = 6$ and $n = 4$ is illustrated in Figure 3-b. Note that in this example, the variable x has the value equal to 24 after the first sum and 10 after the second sum. The concept of variable in this language is different from the concept of variable used by many traditional programming languages. The variable x in Figure 3-b cannot be thought of as a position of computer memory, limited to store a single value at each time, as is the case in programming languages like Pascal, for instance, but it has a value, after the first sum, and another value after the second sum. The history of the calculations and the x values are preserved in the algorithm execution.

A formal programming language has associated to it an **immediate automaton** able to perform a set of sentences belonging to the language. The immediate automaton refers to the "virtual machine" capable of executing the sentences of the formal language the way they are described in the algorithm. It is important to teach the immediate automaton for the student, because he will develop algorithms to this automaton, and not to the real computer. For example, in the Language from the World of Robots the immediate automaton should be seen as a "virtual machine" able to command the robots to walk forward, turn left, etc., and this is different from the set of instructions for a real computer processor. We understand that many of the problems with

programming learning relates to the fact that the student does not develop a suitable mental model of the "machine" to which their algorithms are developed.

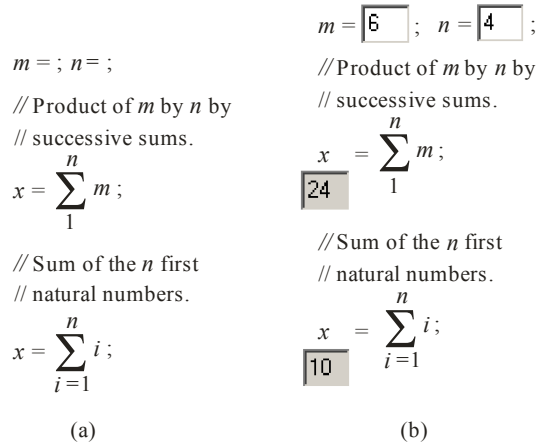


FIGURE 3
LEARNING ENVIRONMENT "*INTEGER AND BOOLEAN ARITHMETIC, ARITHMETIC LANGUAGE*"

FACTORS THAT AFFECT THE PROGRAMMING LEARNING

Space of problems ranges from concrete to abstract. In this paper, we understand the **concrete** as a tendency to perceptible, to the real-visible. **Abstract** refers to what you'll find with a degree of independence for perceptible elements in the world, especially visual references. Abstract is that one considers only existing in the level of ideas, without material basis. A space of problems tends to the concrete when it is formed by elements keeping more direct relationships with perceptible objects and phenomena. Conversely, if it is made up of elements which are abstractions, ideas, it tends to the abstract. The space of problems of the World of Robots is more concrete than that of Mathematics, for example, because it involves concepts of movement that have visual references in the world. It is easier for students to deal with concrete domains because they generally refer to many everyday experiences, perceived and understood, of the real world.

The **size of the problems' domain** also affects the programming learning. Space of problems with thousands of concepts requires from the learner a cognitive effort more than spaces with few concepts. Small problems' domains offer greater comfort for the development of initial programming skills, leaving the learner to focus on the computational field.

If the grammar and semantics of the programming **language matches the problems' domain**, offering primitives that accurately depict the concepts of the space of problems, then the effort of a beginner for writing algorithms tends to be lower, because the solution strategies and reasoning with elements from the space of problems will be most directly expressed by the language structures. Conversely, great effort must be done to translate/simulate the concepts of a domain far from primitives of a programming language. The Language from the World of Robots presents, in its grammar/semantics, primitives such as "GoForward" and "TurnLeft" that match the concepts to move forward and to turn left existent in the space of problems from the World of Robots. On the other hand, it is more difficult to express solutions to problems from the World of Robots using as language, for example, Pascal or Java, since concepts such as "robot", "move forward" and so on, would have to be translated through variables, constants, data types, etc. As another example, consider the space of problems from Integer and Boolean Arithmetic and the languages that we designed to operate in this domain, whose descriptions we made previously. The Arithmetic Language provides primitives which match up more promptly with the concepts in this domain than the Assembly Language from the Simple Computer.

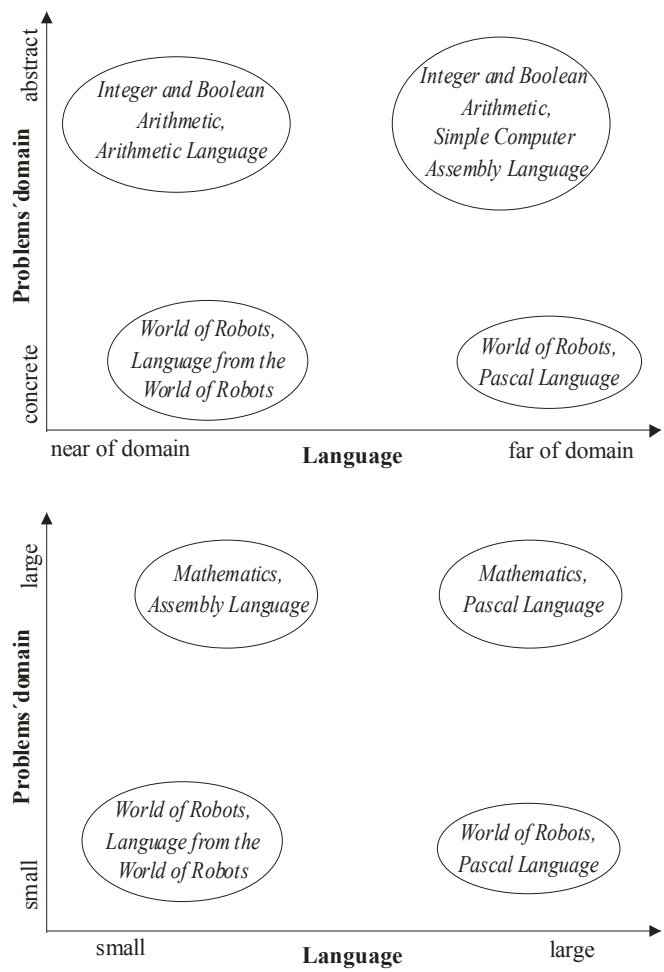


FIGURE 4
LEARING ENVIRONMENTS AND FACTORS THAT AFFECT PROGRAMMING LEARNING

Professional programming languages include productions for exceptions handling, classes declaration, objects instantiation, threads definition, dozens of libraries, among many other linguistic structures, which makes the choice and concentration harder for the new student. Dealing initially with **small languages** enables the learner to focus on strategies for solving of problem and skills needed to develop algorithms.

Offering a huge amount of tools for rapid development of interfaces, components for different applications, support for multiple programming languages and operating systems, support for managing large projects and distribution of software, **professionals programming environments are too complex** to be used by programming students. For the beginner, it is easier to use initially a development environment specifically designed for a certain space of problems and a formal programming language.

We present in Figure 4 a set of learning environments and how they stand in relation to factors affecting the programming learning.

METHODOLOGICAL APPROACH

The central idea of our proposal for an introduction course to programming is in the exposition of the students to different space of problems, each with its own formal language. A sequence of spaces of problems of increasing complexity, will allow the student the gradual assimilation of concepts. Considering the factors that affect the programming learning, earlier described, the learning environment should evolve: (1) from concrete space of problems to abstract space of problems; (2) from space of problems with small number of concepts to large space of problems; (3) from formal language with vocabulary and semantics matched to the problems' domain to generic languages formed by primitives conceptually distant from the space of problems; (4) from formal language with a small vocabulary to languages with grammars containing large amounts of productions.

In addition to this general framework, our proposal is based on the following principles:

- Each "space of problems/formal language" configuration must have a programming environment that integrates specific setting facilities related to editing, compilation, execution, debugging, and visualization of concepts existent in the language;
- The immediate automaton and grammar of each language must be thoroughly explained to students through documents and oral statements;
- The student should practice writing algorithms not only on paper. The use of computers as a means that allows the execution of the solution should be encouraged, creating, thus, a constructivist [12] learning laboratory.

Our proposal can be instantiated by using many different settings of "space of problems/ formal language" according to the educational intentions of courses. For example, a course in Electronic Engineering may want to instantiate a space of problems in the field of digital circuits, while a course in Computer Science with focus on Artificial Intelligence may want to instantiate a language of the Logic paradigm. A possible sequence of settings could include: (1) World of Robots, Language from the World of Robots, (2) Integer and Boolean Arithmetic, Language of Arithmetic, (3) Integer and Boolean Arithmetic, Assembly Language from the Simple Computer, (4) Mathematics, Imperative High Level Language, (5) Open Space of Problems, Language and IDE of the Free Pascal.

RESULTS IN CLASSROOM OF ENGINEERING AND COMPUTER SCIENCE

We experimentally investigate: (1) the students' ability to formulate strategies to solve problems, (2) the correct use of formal language for describing algorithms, specifically the ability of students to use sequential structures, decision and repetition. We use the averages from evaluations by students from four classes of Telecommunication Engineering and Computer Science from Faculty of Campo Limpo Paulista, after completing the initial study of flow control structures for algorithms. The treated group used as a learning environment the World of Robots, while the control group used a traditional approach based on Math problems and use of Pascal language. Table I shows the number n of students involved in each group, the average m and variance s^2 of the students averages.

Each of the four samples showed normal distribution according to normality criteria of D'Agostino test, and the differences fit within the critical values of 1%. The Student t-test was used to compare the results of each pair of "treated/control" group. The performance of the treated groups of Telecommunication Engineering and Computer Science were significantly better ($p < 0.01$) than each of their respective control pairs. The result indicates the effectiveness of the World of Robots as an educational artifact to enable the introductory learning of a formal grammar and use of basic structures for flow control.

TABLE I
EXPERIMENTAL RESULTS

	Treated			Control		
	n	m	s^2	n	m	s^2
Telecomunications Engineering	23	7.1	1.4	27	5.9	2.5
Computer Science	28	7.6	1.2	21	6.3	2.3

CONCLUSIONS AND FUTURE WORK

In this paper we present the results of the first stage of a research that involves a didactic and pedagogic project,

software development, the experimentation and the evaluation of a proposal to facilitate learning of algorithms and computer programming. Our results were presented as: (1) a logical model that explains sources of learning difficulties for developing algorithms, (2) a proposal for a methodology and of a method of teaching-learning for an Introduction to Programming course; (3) experimental data about the effectiveness of the proposal.

Statistical analysis on experimental data indicates the effectiveness of the World of Robots in order to improve the students performance in the learning of grammar and semantics of a formal language and, specifically, the use of flow control structures for algorithms. Future research will focus on measuring the effectiveness of other "space of problems/formal language" settings described in this paper and in the methodology and the method of teaching and learning described. We also intend to expand our universe of observation in order to include classes of Introduction to Programming from other universities.

REFERENCES

[1] Beck, L. L., Chizhik, A. W., An experimental study of cooperative learning in CS1, *ACM SIGCSE Bulletin*, Vol. 40, Nº 1, 2008, pp. 205-209.

[2] Burch, C., Jigsaw, a Programming Environment for Java in CS1, *Journal of Computing Sciences in Colleges*, Vol. 24, Nº 5, 2009, pp. 37-43.

[3] Cilliers, C., Calitz, A., Greyling, J., The effect of integrating an Iconic programming notation into CS1, *ACM SIGCSE Bulletin*, Vol. 37, Nº 3, 2005, pp. 108-112.

[4] Cooper, S., Dann, W., Pausch, R., Using Animated 3-D Graphics to Prepare Novices for CS1, *Computer Science Education*, Vol. 13, Nº 1, 2003, pp. 3-30.

[5] Dann, W., Cooper, S., Alice 3: concrete to abstract, *Communications of the ACM*, Vol. 52, Nº 8, 2009, pp. 27-29.

[6] García-Mateos, G., Fernández-Alemán, J. L., A course on algorithms and data structures using on-line judging, *ACM SIGCSE Bulletin*, Vol. 41, Nº 3, 2009, pp. 45-49.

[7] Jurado, F., Molina, A. I., Redondo, M. A., Ortega, M., Learning to Program with COALA, a Distributed Computer Assisted Environment, *Journal of Universal Computer Science*, Vol. 15, Nº 7, 2009, pp. 1472-1485.

[8] Klassen, M., Visual Approach for Teaching Programming Concepts, *Proceedings of the 9th International Conference on Engineering Education*, San Juan, Puerto Rico, 2006, pp. TIA 1- TIA 6.

[9] Miliszewska, I., Tan, G., Befriending Computer Programming: A Proposed Approach to Teaching Introductory Programming, *Issues in Informing Science and Information Technology*, Vol. 4, 2007, pp. 277-289.

[10] Moskal, B., Lurie, D., Cooper, S., Evaluating the effectiveness of a new instructional approach, *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, Norfolk, Virginia, USA, 2004, pp. 75-79.

[11] Mullins, P. M., Conlon, M., Engaging students in programming fundamentals using Alice 2.0, *Proceedings of the 9th ACM SIGITE Conference on Information Technology Education*, Cincinnati, OH, USA, 2008, pp. 81-88.

[12] Papert, S., *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, 2nd, New York, NY, USA, 1993.

[13] Pattis, R., E., *Karel the Robot: a gentle introduction to the art of programming*, 2ª. ed., New York, John Wiley & Sons, 1995.

[14] Sanders, K., Thomas, L., Checklists for grading object-oriented CS1 programs: concepts and misconceptions, *ACM SIGCSE Bulletin*, Vol. 39, Nº 3, 2007, pp. 166-170.

[15] Stevenson, D. E., Wagner, P. J., Developing real-world programming assignments for CS1, *ACM SIGCSE Bulletin*, Vol. 38, Nº 3, 2006, pp. 158-162.

[16] Summet, J., Kumar, D., O'Hara, K., Walker, D., Ni, L., Blank, D., Balch, T., Personalizing CS1 with robots, *ACM SIGCSE Bulletin*, Vol. 41, Nº 1, 2009, pp. 433-437.

[17] Truong, D., Bancroft, P., Roe, P., ELP - A Web Environment for Learning to Program, *Proceedings of the 19th Annual Conference of the Australasian Society for Computers in Learning and Tertiary Education – ASCILITE´2002*, Auckland, New Zealand, 2002, pp 1-10.

[18] Villalobos, J. A., Calderon, N. A., Jiménez, C. H., Developing programming skills by using interactive learning objects, *ACM SIGCSE Bulletin*, Vol. 41, Nº 3, 2009, pp. 151-155.

[19] Wicentowski, R., Newhall, T., Using image processing projects to teach CS1 topics, *ACM SIGCSE Bulletin*, Vol. 37, Nº 1, 2005, pp. 287-291.