Computers & Education 55 (2010) 767–776

Contents lists available at ScienceDirect

Computers & Education

journal homepage: www.elsevier.com/locate/compedu

Learning computer programming: Implementing a fractal in a Turing Machine

Hernane B. de B. Pereira[a,b,*], Gilney F. Zebende[a,c], Marcelo A. Moret[a,c]

[a] *Programa de Modelagem Computacional, SENAI Cimatec, Av. Orlando Gomes 1845, 41.650-010 Salvador, Bahia, Brazil*
[b] *Departamento de Ciências Exatas, Universidade Estadual de Feira de Santana, Av. Transnordestina, S/N, 44036-900 Feira de Santana, Bahia, Brazil*
[c] *Departamento de Física, Universidade Estadual de Feira de Santana, Av. Transnordestina, S/N, 44036-900 Feira de Santana, Bahia, Brazil*

A R T I C L E I N F O

Article history:
Received 10 November 2009
Accepted 14 March 2010

Keywords:
Applications in subject areas
Cooperative/collaborative learning
Interdisciplinary projects
Programming and programming languages
Teaching/learning strategies

A B S T R A C T

It is common to start a course on computer programming logic by teaching the algorithm concept from the point of view of natural languages, but in a schematic way. In this sense we note that the students have difficulties in understanding and implementation of the problems proposed by the teacher. The main idea of this paper is to show that the logical reasoning of computer programming students can be efficiently developed by using at the same time Turing Machine, cellular automata (Wolfram rule) and fractals theory via Problem-Based Learning (PBL). The results indicate that this approach is useful, but the teacher needs introducing, in an interdisciplinary context, the simple theory of cellular automata and the fractals before the problem implementation.

1. Introduction

The main idea of this paper is to present a strategy for learning of computer programming in interdisciplinary classes. This strategy began with the assumption of how not to make boring teaching and at the same time inducing connections among the different areas of knowledge. In quest of that goal, we introduce the learning method based on problems (Section 4.1) with the main objective of the resolution of an interdisciplinary problem that involved Logic, computer programming language, cellular automata and fractals.

The proposed hypothesis is that a reduction in the repertoire of commands to solve an algorithmic problem was more stimulating for students throughout the process of solving the problem. But, there is a worry question: among the programming languages, what is it that “restricts” the basic set of commands? The response is to be found in abstract machines such as the Turing Machine and the Post Machine.

We have a proposition in Tenório (1991) chapter 4 that is the kernel of this paper: “How does the work of Turing and Post, in the same way that it happens in computer electronics, influence or may affect the relations of knowledge production, especially the relations of the subjects with the knowledge to be (re) produced in the school?”

Considering that the digital computer is based on the binary system, we believe that the inclusion of experiments that encourage students to think in a similar way to computers can facilitate their learning with respect to computer programming. In this paper, we present some reflections on the learning of computer programming, resulting from a teaching experiment that takes into account the implementation of a cellular automaton, Wolfram rule 90, in the Turing Machine. The goal here is to show that in a simple way, through an abstract machine and with a limited repertoire of instructions, we can more effectively develop the logical thinking of students of computer programming.

This paper is organized in the following way. Section 2 presents an overview of the Turing machine, so that it gives the reader a formal definition, the basic concepts and operation. Section 3 gives a brief introduction on the cellular automata as the basis for understanding the experiment. Section 4 describes the experiment, showing the method of teaching–learning used and detailing the experimental stages. Section 5 present the results obtained from a questionnaire and observations. Finally, in Section 6 we present our concluding remarks.

2. The Turing Machine

In 1936, the British mathematician, Alan Mathison Turing, many years before any modern digital computers existed, conceived of an abstract model of a computer, i.e. the conceptual structure, which can be used to represent algorithms. This abstract machine, which came to be known as the Turing machine (TM), covers just the logical aspects of its operation and not its physical implementation.

* Corresponding author. Programa de Modelagem Computacional, SENAI Cimatec, Av. Orlando Gomes 1845, 41.650-010, Salvador, Bahia, Brazil.
E-mail addresses: hbbpereira@gmail.com (H. B. de B. Pereira), gfzebende@hotmail.com (G. F. Zebende), mamoret@gmail.com (M. A. Moret).

0360-1315/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compedu.2010.03.009

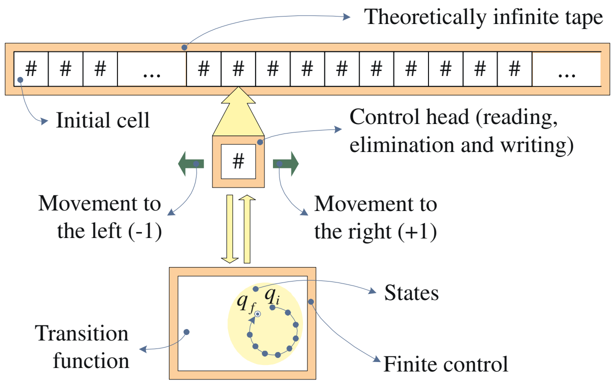
Theoretically infinite tape
# ... # # # # # # # # # # ...
Initial cell
Control head (reading, elimination and writing)
#
Movement to the left (-1)
Movement to the right (+1)
q_f q_i
States
Transition function
Finite control

Fig. 1. Conceptual scheme of the Turing Machine.

This and others of a similar type (e.g. Emil L. Post's machine, better known as the Post Machine) enabled the development of computers – from the perspective of logic – in that they deal with, accurately, the concept of computability. This concept is strongly related to the definition of algorithms. Svaiter (2008), when presenting an introductory overview on complexity in computing, argues that:

"We can create an imaginary computer with unlimited memory and ask ourselves what are the problems that computer can solve, without worrying about the total time spent. (...) The tasks that can be performed by these computers are (problems or functions) computable."

It is clear that a TM can model any digital computer. However, it is easier to explain what a TM is by presenting it as a machine in a physical sense. In this context, we can observe that these abstract models make it possible to set limitations for mechanical procedures. For example, in the literature (e.g. Aho, Hopcroft, & Ullman, 1974; Du & Ker-I, 2001), we find studies in which variations of TM (e.g. a strip of k-tape, etc.) are discussed. Following on, we briefly describe the simplest of variations of TM that exist, i.e. the TM of a tape, presenting its features, its formal definition and its operation.

The physical component of a TM is made up of three parts: (1) a theoretically infinite tape divided into cells or houses, with a symbol written in each cell, (2) a control head that moves along the tape, carrying out one or other of two possible moves (i.e. left or right), stopping at each house and performing the operations of reading and elimination of the contents of a cell and writing new content, which could be rewritten and (3) a finite control that manages the head above and contains a finite set of instructions that will be implemented during the program generating new states at each instant $t+1$ (Fig. 1). Moreover, according to various authors, e.g. Aho et al. (1974), Campello and Maculan (1994), Du and Ker-I (2001) and Tenório (1991), this machine or abstract model, which performs basic functions, can be described by a tuple of 5–8 components. In this paper we have a tuple of 5 components $(Q, \Sigma, \Gamma, \delta, q_i)$ (Table 1).

Besides these components, there is the final state represented by q_f, that here is assumed not to be part of Q ($q_f \notin Q$) and the white-space character that are both common to the TM. It is common to find a variation in terms of number of components that describe the machine (e.g. Campello & Maculan, 1994; Hopcroft, Motwani, & Ullman, 2002, among others).

There are many ways to conceive of the concrete operation of a TM, and therefore to structure its instructions. In general, a TM is deterministic, i.e. for every situation there is only one instruction that the machine has to follow. There are also non deterministic TM, but these machines are not described in this paper. For simplicity, we considered only the characters '0' and '1' in the TM. Fig. 2 shows two examples of notations for a typical instruction of a TM.

The set of instructions is the program that is stored in "memory" of the machine. When the machine stops, it displays a result. If a TM does not stop during the implementation of a program, it is likely that there is a defect or a possible lack of a decision (at least in finite time) in relation to the problem.

Thus, using the description of the TM presented above as a starting point, we observed that to formalize the procedure of a TM it is necessary to develop a notation for the settings or according to Hopcroft et al. (2002), instant descriptions.

In this paper, we use transition diagrams, which allow the representation of the transitions of a TM using a model similar to a graph. The states are the nodes and the movements of each state are the edges. For didactic reasons and to avoid this paper becoming too long, the exemplification of the formal procedure and the transition diagram is presented in Section 4.2, where a proposal for a result to the problem of the experiment is commented on.

Table 1
Description of TM components.

Components	Description
Q	A finite set of states.
Σ	A finite set of input alphabet not containing the white-space character (here represented by the character '#', although one can find other representations such as '_', 'B', 'Δ').
Γ	A finite set of tape alphabet with $\Sigma \cup \{\#\} \subseteq \Gamma$
δ	The state-transition function, i.e. instructions that describe the actions that will be carried out during the execution of a program.
q_i	The initial state.