Towards a definition of the concept of logic of an algorithm

Federico Gómez
Instituto de Computación -
Facultad de Ingeniería
Universidad de la República
fgfrois@fing.edu.uy

Sylvia da Rosa
Instituto de Computación
Facultad de Ingeniería
Universidad de la República
darosa@fing.edu.uy

Abstract

Within the framework of an empirical study carried out with students of an introductory programming course of a Computer Engineering Under-graduate Program, the students were asked to design an algorithm to solve the linear search problem and implement the solution, using two imperative programming languages (Pascal and C++) in two different groups of students. It was found that the solutions proposed by students followed different strategies (different "logics") to solve the problem. This fact led to the need to propose a definition of the concept of logic of the algorithm, as far as we know, not found in the literature, as well as to explore the reasons that lead each student to follow one strategy or another.

This article describes the activities carried out during the study and includes selected excerpts of students' responses. The results are analyzed in the framework of the theory of the investigation (Genetic Epistemology of Jean Piaget) with the focus on students solutions according to the proposed definition of logic of the algorithm. Conclusions and some future work that may eventually lead to a more exact definition are included.

1. Introduction

In 2022, we carried out an empirical study that investigates the construction of knowledge about the linear search problem, its algorithmic solution, and the implementation and execution of a program that solves it on a computer (da Rosa & Gómez, 2022). The study had as the main goal to investigate the process of students' thinking from solving a concrete instance of the problem to the writing of a program that implemented the search for a given value in an array of integers. The students were asked to explain how and why the program worked when executed on a computer, both at the level of its source code (the textual part of the program) and its execution on a machine (the executable part of the program). The dual nature of a program (textual and executable) was disscused in (da Rosa,S. & Chmiel,A. & Gómez, F., 2016). As for other empirical studies in which we investigate knowledge construction of several computing concepts, the study is based on principles of Jean Piaget's Genetic Epistemology theory (Piaget, 1977; Piaget & Garcia, 1980). In (da Rosa & Aguirre, 2018) the general law of cognition that Piaget describes to explain the construction of algorithmic knowledge (Piaget, 1964), is extended to encompass the construction of computational knowledge (da Rosa, 2018). The extended law of cognition is used in detail in (da Rosa & Gómez, 2022) to explain the process of students' thinking from solving the concrete instance of the problem (instrumental knowledge), to explaining their algorithms (conceptual knowledge) and writing the general program (formal knowledge). Besides the expected results, we observed an additional fact that we had not initially foreseen in the study design: when writing down the steps of the algorithm that solved the problem, some students used a certain strategy to implement the search, while others employed a different strategy. In order to easily identify them in the context of the study, we decided to name them the *a priori* strategy and the *a posteriori* strategy (see Section 3. Each strategy follows its own "logic", in the sense that it executes the steps that solve the problem in a different order than the other strategy. This made us formulate two investigation questions:

- First, what are the reasons that led each student to propose one or the other when implementing their solution, given that both the instructions and the activities proposed in the study are the same for all students.
- Second, whether there is a formal definition of the concept of "logic" of an algorithm, as the

term is often used informally to refer to the behavior given by the steps that define its algorithmic strategy and distinguish it from other strategies (other "logics").

The rest of this paper is organised as follows: Section 2 presents a summary of the empirical study on the linear search. Section 3 analyses possible responses to the questions above. Finally, in section 4, some conclusions are presented, as well as some lines of future work that could lead to a more precise definition of the concept of logic of an algorithm. Finally, references are included.

2. Summary of the empirical study

The first part of the study focuses on the process of students' thinking from solving a concrete instance of the problem of linear search to the writing of an explanation about how they did and why their methods solves the problem. According to Piaget's general law of cognition (Piaget, 1964), the success of students in doing that is evidence of the construction of conceptual knowledge about the algorithm.

The second part of the study examines the process of students writing a program from their explanations and executing it on a computer, which according to the extension of the aforementioned law is evidence of the construction of formal knowledge (da Rosa & Aguirre, 2018).

Thirteen students from an introductory programming course participated in the study: seven students working with C++ and six students working with Pascal. The topics covered in the courses were identical for both groups, the only difference was the programming language used. All activities were carried out individually by the students.

The concrete instance of the linear search problem is presented to the students in the form of searching for ID numbers in a row of numbered cards representing door numbers of houses on a street, as shown in Figure 1. Each card has a second card underneath, representing the ID number of the person living in that house, as shown in Figure 2. It is assumed that only one person lives in each house. The ID numbers are not ordered and hidden from the student's view. The student is asked to search for a number that is in the row and for another one that is not.

Figure 1 – Simulation with houses

Figure 2 – Simulation with cards

Students successfully solve the problem working with the row of cards and answer questions aimed at obtaining accurate descriptions of the method used and why they were successful, providing a written description in natural language. As an example, the description in natural language given by one of the students is shown below. *Assuming that you want to find the person with ID number x, you go through the doors in order, opening them and asking the person behind them for their ID number. If it's the one we were looking for, the search ends, otherwise, we go to the next door. If the doors are visited and the person with ID number x is not found, it is deduced that he/she does not reside behind any of the doors.*

The student expresses all necessary actions for solving the problem, comparison (*asking the person behind them for their ID number. If it's the one we were looking for*), advance (*we go to the next*

door) and repetition (implicitly in the description of actions referring to the doors, in plural). He also describes the changes imposed on objects, he stops both when he finds the ID number (*If it's the one we were looking for, the search ends*) and when there are no more doors to visit (*it is deduced that he/she does not reside behind any of the doors*) (da Rosa & Gómez, 2022).

Once all students write similar descriptions, they are asked to write a version of the algorithm using pseudocode, based on her/his description in natural language. Pseudocode allows an external agent (in this case, an imaginary robot played by the interviewer) to execute it, which helps the student visualise the behaviour of the algorithm and correct errors, leaving for later aspects related to machine execution. This tool is called **automation** and induces the student to reflect on errors that may be detected and correct them. The student is asked to write multiple progressive versions of the algorithm until finally arriving at a correct one.

An example of a process with several versions is shown below.

```
while (not find id number)
    ask for id number in door
    if it's the one I look for
        stop
    else
        end search
    end
end
```

In this version, the student expresses two of the three actions: comparison (if it's the one I look for) and repetition (while), while omitting the action of advancing to the next door (he writes "end search" after else). Regarding the condition for the while loop, the student intends for the external agent to stop searching upon finding the desired number, but does not consider the case where there are no more doors remaining. The student's primary focus lies on achieving the desired result. He has conceptualised the algorithm after executing it by himself but needs to further conceptualise the action to be taken when the number is not found behind the current door. In this case a second condition for stopping the iteration when there are no more doors left has to be included. This student needs to write three more versions before finally coming up with a correct one. Between each version and the next, automation was used to detect and fix various errors. The final (and correct) version of this student's pseudocode is shown below.

```
while (not find id number) and (there are more doors)
    ask for id number in door
    if it's the one I look for
        stop
    else
        go to the next door
    end
end
```

Given that the goal is to write and execute on a computer a solution to the more general problem of searching for a value in an array with N cells, each student first writes some code snippets to gain familiarity with the syntax and semantic rules associated with arrays in her/his respective programming language (C++ or Pascal) before continuing with the writing of the program.

In the final activity, each student writes, compiles and executes a program that searches for a given value in an array of N integers, based on their previous pseudocode. The focus is on the correspondence between the steps of the pseudocode and the program instructions (knowledge about the *textual* part of the program) and on aspects specific to machine execution (knowledge about the *executable* part of the program) (see Section 1).

Automation is used to detect and correct errors, using a paper-based array to visualise the execution, guiding the student to consider machine execution issues, such as an infinite loop or an out-of-range index error.

As with the pseudocode, each student needs to write more than one version of the program prior to arriving at a correct one. As an example, the final C++ version of the same student included above is shown below.

```
boolean found = FALSE;
int i = 0;
while ((found == FALSE) && (i <= N-1)) {
    if (arre[i] == number) {
        found = TRUE;
    } else {
        i = i+1;
    }
}
```

The student establishes a suitable correspondence between the steps in pseudocode and the instructions in the program. At the end of the study, all the students (both those who worked with C++ and those who worked with Pascal) successfully wrote a correct program, compiled and executed it on a computer.

3. Different logics within the implemented algorithms

Analysing students' algorithms using pseudocode, we found that the final version written by each student follows one (and only one) of the following search strategies, that we call *a priori* and *a posteriori*.

- A priori: the student consults the ID number at the first door before starting the iteration. If the searched ID number is not in the row, she/he ends up positioned at the last door at the end of the search.
- A posteriori: the student consults the document at the first door after starting the iteration (i.e., within the iteration itself). If the searched document is not in the row, she/he advances once more after consulting the last door and only then finishes the search.

Five students follow the *a priori* strategy in their final version, while the remaining eight follow the *a posteriori* strategy. In some cases, the employed strategy is already noticeable in the initial version, while in others, it becomes evident as their versions progress. When designing the study, no specific consideration is given to either of the two strategies. The arrival of each student at one of them is observed during the implementation. The students who follow the *a priori* strategy never position themselves beyond the last door. Whether the searched document is in the last door or it doesn't exist in the row, they are equally positioned at the last door, with the iteration stopping based on either one of the conditions of the while loop depending on whether the document is found or not. For example, the following is the implementation of this strategy by student 1:

```
Check ID at Door 1
While (there are more Doors) AND NOT (it is the ID I'm looking for)
    Check ID at next Door
End
```

The students who follow the *a posteriori* strategy advance once more after visiting the last door when the searched document is not in the row. However, they appropriately control the termination, avoiding checking for the document at a door that doesn't exist. For example, student 2 follows this strategy in his final version:

```
    While (there are doors in the block) AND (haven't found the person)
    Knock on the door
    Ask for the person's ID
    If (it is the person I'm looking for)
        Finish the search
    Else
        Go to the next door
    End
End
```

Informally, we refer to the algorithm "logic" as the order of execution of the steps expressed in pseudocode, without yet considering aspects of their subsequent implementation in a programming language. This involves, for example, the choice of a data structure to be used in the program. In the study, linear search on an array is chosen for implementation, but it could also be done on a linked list. In both implementations, the data structure varies, but the algorithm logic remains the same. What determines the algorithm logic is the behaviour produced when the external agent executes the pseudocode steps in the defined order. However, the observation of both strategies is something that drew our attention, as all students were given the same instructions and the study design was never intended to induce one strategy or the other.

In the program writing phase, each student maintains the logic expressed in their pseudocode algorithm, as shown below.

- Student 1 (Pascal language, *a priori* strategy, array indices ranging from 1 to N)

```
door := 1;
WHILE (Door <= N) AND NOT (id = arre[Door]) DO
    door := door + 1;
```

- Student 2 (C++ language, *a posteriori* strategy, array indices ranging from 0 to N-1).

```
boolean found = FALSE;
int i = 0;
while ((i ≤ N-1) && (!found))
{
    if (arre[i] == ID)
        found = TRUE;
    else
        i++;
}
```

It can be observed that the *a priori* programs do not use selection (if/else) or boolean variables, whereas the *a posteriori* programs do use both elements.

A remarkable fact is that almost all students working with Pascal follow the *a priori* strategy (only one follows the a posteriori strategy) and all the students who used C++ follow the *a posteriori* strategy. However, the definition of the search strategy emerged when specifying the logic of the algorithm during the **pseudocode** writing phase where the thirteen students used the exact same pseudocode syntax rules, **prior** to the writing of the code in the programming language. Even more, both strategies can be implemented in both languages, which allow working with the same elements (integer and boolean variables, if/else and while structures).

This led us to wonder if prior knowledge of the formal language could influence the student's thinking when constructing knowledge about the logic of a new algorithm before its formalisation. At the beginning of the study, each student only knew the language used in their group, and none of them had

worked with arrays before. They had all worked with the same elements (variables, basic data types, expressions and simple instructions, selection structures, and iteration structures).

To find an explanation for the observations, we reviewed the curricula of both groups and found a single difference that may be relevant. In the Pascal course much emphasis is placed on the difference between short-circuit and complete-circuit evaluation of boolean expressions, because part of the bibliography is a book describing standard Pascal, that evaluates boolean expressions with complete-circuit, while the compiler used in the course is the Free Pascal compiler that evaluates boolean expressions using short-circuit. Because of that, the students are encouraged to evaluate using short-circuit, ignoring that part of the course book. The C++ group works with Code::Blocks and although the difference between complete and short circuit as well as the fact that Code::Blocks evaluates by default using short-circuit were known, no especial emphasis is placed on the topic. Therefore, students in C++ may unconsciously assume complete-circuit, tending to use boolean variables and if/else statements while Pascal students may have a greater awareness of short-circuit, avoiding the use of boolean variables and if/else statements. These observations are an example of the impact that prior knowledge constructed in different domains has into mental schemas and how it becomes integrated in the process of construction of new concepts (Celleriér, 1987).

In the context of the study, although students define the algorithm logic in the pseudocode stage, they had become acquainted with short-circuit and complete-circuit evaluation. When defining the logic in pseudocode the Pascal group integrated the short-circuit into their mental schema of the new concept, due to the emphasis mentioned above. Anyway, these are primary explanations that deserve to be deepened through future studies, in the same way that the expression *logic of the algorithm* deserves a formal definition, as posed as the investigation questions in Section 1.

4. Conclusions and further work

Within the framework of the study, we propose defining the concept of *logic of an algorithm* as the order of execution of the steps expressed in pseudocode, before considering aspects of its implementation in a programming language. We consider that this definition is insufficient because it does not express the concept with the desired accuracy. The logic of an algorithm concerns not only the order of the execution of the steps, but there is something underlying that leads a person to propose one order over another when devising the steps of the algorithm. We found an explanation related to what was observed in this study with the *a priori* and *a posteriori* search strategies.

However, we believe that it is necessary to further investigate how individuals construct the logic of an algorithm prior to its formalization in general, which could be linked to the logic of actions and significations developed by Piaget and García in (Piaget & García., 1987). These investigations would complete a cycle of our work that focuses on studying the construction of knowledge about algorithms, data structures, and programs. In this context, understanding the construction of the logic of an algorithm is fundamental since it permeates the entire construction process, from its genesis at an instrumental level, through the conceptual level, and reaching the formal level. In all the studies conducted, it was observed that this construction is complex. The evidence collected shows that previously constructed schemas influence this construction (Celleriér, 1987), including formal knowledge, as observed in the study with the evaluation of boolean expressions using short-circuit or complete-circuit.

During this work, we reviewed the academic literature for any definition of the concept (even under other denominations that do not involve the word "logic") that aligned with the notion proposed in Section 3, but we did not find any. The closest we found is Kowalski's proposal in (Kowalski, 1979): "An algorithm can be regarded as consisting of a logic component, which specifies the knowledge to be used in solving problems, and a control component, which determines the problem-solving strategies by means of which that knowledge is used." However, this proposal does not reflect the same notion that we use, because it associates the idea of logic with specification rather than behaviour, associating the latter with what he calls control. Furthermore, in the rest of the work, Kowalski links both notions (logic and control) with the data structures used in the implementation, deviating from the idea that the logic of an algorithm

does not depend on the structure on which it is subsequently implemented in a programming language as mentioned in Section 3) for the case of the algorithm that solves the linear search, that has the same logic whether implemented on an array or a linked list.

The contrast between the difficulty in finding a definition of the concept of logic of an algorithm and its underlying presence in all the studies conducted during the elaboration of the model for investigating the construction of knowledge of programs (da Rosa & Gómez, 2019) leads us to think that obtaining empirical data may eventually lead to the emergence of a more precise definition. In our opinion, this is closely linked to the absence of a more precise definition of the concept of computational thinking. Since Wing popularized a characterization of it involving solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science (Wing, 2008), the literature has proposed multiple attempts to define the computational thinking, all of which are incomplete, as mentioned by da Rosa in (da Rosa, 2018). In the same work, da Rosa suggests that an adequate definition of the concept could arise from an extension of the general law of cognition defined in the same paper to explain the construction of formal knowledge. It is believed that delving deeper into this line of work could provide more precise and complete definitions for both concepts, while also allowing for a deeper explanation of how individuals construct knowledge about algorithms, data structures, and programs.

5. References

Celleriér, G. (1987). *Structures and Functions in Piaget today.* Lawrence Erlbaum Associates Publishers.

da Rosa,S. & Chmiel,A. & Gómez, F. (2016). Philosophy of Computer Science and its Effect on Education - Towards the Construction of an Interdisciplinary Group. *Special edition of the CLEI Electronic Journal (see http://www.clei.cl/cleiej/), Volume 19 : Number 1 : Paper 5*.

da Rosa, S. (2018). Piaget and Computational Thinking. *CSERC '18: Proceedings of the 7th Computer Science Education Research Conference*, 44–50. https://doi.org/10.1145/3289406.3289412.

da Rosa, S., & Aguirre, A. (2018). Students teach a computer how to play a game. *LNCS of The 11th International Conference on Informatics in Schools ISSEP 2018* .

da Rosa, S., & Gómez, F. (2019). Towards a research model in programming didactics. *Proceedings of 2019 XLV Latin American Computing Conference (CLEI)*, 1–8. doi: 10.1109/CLEI47609.2019

da Rosa, S., & Gómez, F. (2022). The construction of knowledge about programs. *Proceedings of PPIG 2022 - 33rd Annual Workshop*, 1–8.

Kowalski, R. (1979). Algorithm = Logic + Control. *Communications of the ACM, vol 22, nº 7*, 424–436.

Piaget, J. (1964). *La prise de conscience*. Presses Universitaires de France.

Piaget, J. (1977). Genetic Epistemology, a series of lectures delivered by Piaget at Columbia University, translated by Eleanor Duckworth. *Columbia University Press*.

Piaget, J., & Garcia, R. (1980). *Psychogenesis and the History of Sciences*. Columbia University Press, New York.

Piaget, J., & García., R. (1987). *Hacia una lógica de significaciones*. Gedisa, ISBN 9784743266640.

Wing, J. (2008). Computational thinking and thinking about computing. *Philosophical transitions of the Royal Society*, *Phil. Trans. R. Soc. A 366,*, 3717–3725.