Learning (through) Recursion: A Multidimensional Analysis of the Competences Achieved by CS1 Students

Claudio Mirolo
Dept. of Mathematics and Computer Science – University of Udine
via delle Scienze, 206 – 33100 Udine (Italy)
claudio.mirolo@dimi.uniud.it

ABSTRACT
In this paper I will discuss an investigation intended to address the learning of recursion in a multidimensional perspective, where the dimensions correspond to different types of competence relevant to programming. One such dimension is the understanding of the computation model, that I have assessed under the methodology proposed by Göttschi et al. (2003). Moreover, I have tried to analyze and correlate other learning dimensions, such as the ability to establish relations in the problem domain, to deal with recursive structures, as well as to develop basic abstraction skills. One of my objectives is indeed to gain a better understanding of the major sources of difficulties that students face. In essence, my investigation lends further support to previous related findings on mental models. However, a consistent model of recursive computations, although implied by the ability to use recursion in problem-solving, does not seem to be sufficient for the achievement of higher-level skills.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education — computer science education

General Terms
Human Factors

Keywords
programming learning, recursion, mental models.

1. INTRODUCTION
It is still generally agreed that “Computer science educators have found that recursion is a very difficult concept for students to learn and teachers to teach” [18]. This has motivated a persistent research interest in several aspects of the topic, e.g. [4, 11, 13, 14], including the comparison of the tasks performed by expert and novice programmers, the classification of pupils’ mental models and misconceptions, the analysis of their descriptions of recursive structures and phenomena. Other authors have suggested new methods for teaching recursion, e.g [1, 2, 6], possibly with the support of specific software aids.

As a teacher of an introductory programming course based on a functional-first approach [3], I am especially interested in understanding the main sources of difficulties that the learners face before making the conceptual leap that would allow them to successfully integrate recursion in their problem solving equipment. In this respect, I like to see the main aspects of introductory programming, including recursion, as involving a multidimensional range of competences, in particular:

i. Learning the formalization medium (language notation);
ii. Understanding the underlying computation model (mechanics of the notional machine);
iii. Reasoning and establishing relations in the problem domain (basic problem solving skills);
iv. Interpreting and exploiting the program structures (basic organizational skills);
v. Conceiving and building abstractions (higher-level organizational skills, management of complexity).

The competence areas (i), (ii) and (iv) were already mentioned in [5], in connection with some difficulties of novice programmers (the terms in italic are borrowed from the cited paper), whereas the need to strengthen the abilities (iii) and (v) is given particular emphasis, e.g., in [17]. From now on, I will refer to the items (i–v) as learning dimensions.

Although a variety of works emphasize the concerns with regard to the computation process, e.g., [13, 8, 10], I am inclined to think that dexterity with a computation model is not per se the crucial achievement that could enable the learners to devise recursive solutions. Understanding the computation process is helpful in that the students feel more comfortable with the fact that a recursive procedure can effectively be computed, as well as to get a more concrete idea of which recursive definitions are well-founded. Beyond this, as warned in [9, 17], the computation model is of little help to approach a problem recursively and should be downplayed in favor of a more declarative and/or abstract approach.

To gain a better understanding of the students’ actual achievements, their fluency with the mechanics of computation should be correlated to other learning dimensions. Hence, I have designed and administered a questionnaire whose structure is conceived precisely to investigate these aspects. In summary, the main research questions are: If the knowledge of the computation model is analyzed under Göttschi’s and colleagues methodology [10, 16], how do their...
findings generalize to a different context? Is the development of a sound computation model predictive of the achievement of other higher-level competences? Is it possible to identify specific learning obstacles to understand recursion?

The rest of the paper is organized as follows. Sect. 2 outlines the questionnaire structure and its rationale. Sect. 3 draws the context and implementation of the experiment. Sect. 4 addresses the methodology and the results of the analysis, which are then briefly discussed in sect. 5.

2. INSTRUMENT

In the attempt to address each of the five learning dimensions introduced above, I have prepared a questionnaire of 11 items, here labeled Q1–Q11, whose general structure can be outlined as follows (the number, order and partition of the items accommodates for some implementation constraints):

Part I
- Dimension (ii): computation model
  Q1 - Trace of a computation: linear recursion
  Q2 - Trace of a computation: tree recursion
- Dimension (iii): relations in the problem domain
  Q3 - Relations between different problem instances: simple procedure application
  Q4 - Relations between different problem instances: application subject to a condition
  Q5 - Relations between different problem instances: application within a compound expression
- Dimension (iv): recursive structures
  Q6 - Analysis: informal description
  Q7 - Synthesis: procedure definition

Part II
- Dimension (i): language structures
  Q8 - Consistent assembly of language constructs
  Q9 - Translation from a different notation
- Dimension (v): abstraction
  Q10 - Procedural abstraction: informal description of functional relations and procedure naming
  Q11 - Code re-organization: abstraction over patterns and processes

Under a functional-first approach, all the considered competences pertain the introductory path since the early steps. The main challenge was then to find suitable questions to address the learning dimensions independently of each other.

2.1 Questions

The purpose of the first two items is to infer the students’ mental models of recursive computations as proposed in [10]. Both Q1 and Q2 ask to trace the key computation steps of a procedure application, including all recursive calls. In particular, the computation implied by Q2 is tree-recursive.

Questions Q3 through Q5 address the relations between different instances of a problem, at the core of recursion, freed from the overload of a full recursive structure. To this aim, I have imagined a sort of game, that is exemplified as follows (the number, order and partition of the items accommodates for some implementation constraints):

\[
\text{if } (> \text{ y } \text{ x}) 1 \ (	ext{let } (\text{ if } (> \text{ y } \text{ x}) 1 \ (	ext{lambda } (\text{x}) \text{ y}))) \text{ y}\]

The answer question Q9, on the other hand, the students have to translate in Scheme the recursive definition of a function, given in mathematical notation.

Q10 presents a larger string-processing program for adding two nonnegative integers, where addends and result are represented by binary strings. Besides a main procedure add, the program consists of five procedures with generic names (P, Q, . . . ), whose roles have to be identified and explained in functional terms, i.e. in terms of “the relations between the returned value and the input arguments.” Moreover, for each such procedure a meaningful name should be chosen. Finally, item Q11 asks to improve the readability of a given program by using let* (abstraction over values) and by defining suitable sub-procedures (procedural abstraction).

The full text of the questionnaire is available online at [http://nid.dimi.uniud.it/download/rec_test_09.pdf](http://nid.dimi.uniud.it/download/rec_test_09.pdf)

2.2 Rationale

Whereas the rationale of Q1–Q2 is drawn from [10, 16], most of the other questions have been conceived to allow two kinds of answers, which may help to distinguish deeper levels of understanding from a more stereotyped learning style.

The solutions to items Q3–Q5 may be just logically correct or also appropriate for a recursive definition, i.e., such that the given procedure is applied to intuitively simplier arguments. To reduce the mathematical demands, these problems are strongly connected to the material presented in class. In the case of Q6, for example, a tail-recursive \textit{gcd} algorithm was already known. Moreover, there are three similar items of this kind in order to discriminate between persistent and occasional difficulties experienced by students.

A neat characterization of the Sierpinski’s triangle in Q6 describes its structure as recursively built up from three resized and displaced instances of lower recursion depth. On the other hand, the same structure may also be seen as the result of (iteratively!) decomposing all the smallest triangles in one instance of lower recursion depth. Similarly, a smart solution of Q7 does not scan the characters sequentially, but tries to match the leftmost and rightmost parentheses and then recursively processes the inner part of the string.

Two different assemblies of the pieces of code provided in Q8 are syntactically correct. However, the more stereotypical procedure definition has a serious drawback, since it is apparent that it gives always rise to infinite recurrences.
A major concern with Q10 is the dichotomy “what to” vs. “how to” [17]. In this case it is crucial the ability to provide functional characterizations of the recursive procedures, that indicates a higher level of abstraction, whereas the task is easier for the other procedures that are based on simple expressions with standard operations. In the last question, to conclude, a nice solution identifies useful sub-tasks and defines the appropriate procedural abstractions, besides applying let constructs to restructure the code.

3. EXPERIMENT

3.1 Context

Almost all the students enrolled in the first year of the undergraduate program in CS at the University of Udine have taken part in the experiment. The complete questionnaire has been assigned during the first semester 2009-10 of introductory programming, a mandatory course, after a previous administration, restricted to part I, in the a.y. 2008-09.

As usual in Italy, first-year students are heterogeneous in terms of their background. Basic high school maths is the only entry requirement for CS, and as many as 30% of them were not exposed to computer programming in the secondary school (although just about 15% assert that they have had no previous programming experience at all).

In the first semester the students learn basic concepts of functional programming and apply them to write recursive procedures in Scheme. The material presented before the test is roughly that covered in the first part of Hailperin’s et al. textbook [12], on procedural abstraction. For most students it is the first real experience with recursion, since in the secondary school they are ordinarily exposed to the imperative paradigm. During the lessons there are few occasions when the students can see examples of application of the typical computation model, based on substitutions and reductions, to evaluate Scheme expressions. However, they can continue to explore this process autonomously, with the aid of the tools available in the DrScheme environment [7].

Most of the time in class and in the labs is devoted to analyzing and solving simple programming problems. Following Wu et al. [18], in order to understand recursion the students are provided with both concrete computation models, as well as abstract mathematical models, notably some proofs of program correctness via mathematical induction.

3.2 Implementation

Each part of the questionnaire was administered to the students in a two-hour session in a controlled situation. The students who took part in the second session, a week later, were also asked individual questions, in order to clarify ambiguities, to find and correct minor mistakes, or to provide more details on their answers to part I. (Also these specific items were collected in written sheets.)

50 students participated in the first session of the a.y. ’09-10 and 38 in the second session (both held in december 2009). In the a.y. ’08-09 the first part of the questionnaire was assigned to 45 students (dec. 2008), whereas the second session was entirely devoted to the individual questions.

It may be worth noting that the students didn’t know anything in advance about the types of questions proposed in the tests, that were indeed unusual if compared to those of standard exams. Nor were they informed on the possibility of answering individual questions in the second session. To attain a balance between “exam anxiety,” on the one hand, and weakening of motivation of the other, the students were told that only the positive clues in their responses would be considered for the assessment of the final exam.

4. ANALYSIS

Before presenting the analysis of students’ answers to the questionnaire, I summarize the approach of Götschi and colleagues [10], which is relevant for interpreting the traces of recursive computations.

4.1 Mental Models

By considering the active flow (descending flow of recursive invocations), the passive flow (ascending flow of suspended operations) and the treatment of the base cases in the students’ traces, Götschi et al. [10] extend Kahn’s taxonomy [13] and identify the following main mental models: Copies model. A mental model in this class provides an accurate account of the active flow, the passive flow, as well as the switches from active to passive flow at each base case. Active model. This kind of incomplete models only support a consistent characterization of the active flow, whereas the passive flow remains vaguely (if at all) sketched. Odd models. Useless models giving rise to hardly comprehensible traces.

Other kinds of infrequent models that indicate important misconceptions include the looping, magic, return value, step and algebraic models, that are described in [10].

4.2 Methodology

Relative to the first two items, for the sake of comparison I have essayed to infer the mental models explaining students’ traces under the methodology suggested by Götschi and colleagues. More specifically, suitable labels have been assigned to the treatments of the active flow, the passive flow and the base cases; then each triplet of labels has been interpreted in terms of mental models (see [10]).

Since some of the resulting categories only apply to sporadic instances, if any at all (this was also observed in later studies of the cited authors [16]), I prefer to consider few broader classes, namely:

- Sound model: corresponding to the copies model;
- Fragile model: introduced in order to account for some traces where parts of the active flow, of the passive flow and of the base cases are well developed, but the occurrence of errors or inexplicable flow jumps may indicate that the mental model is still a little unsteady;
- Forward model: essentially the active model;
- Odd model: as in [10];
- Other: other occasional models, including the looping model, magic model, step model, algebraic model, etc., usually indicating serious misunderstandings.

In accordance with the underlying rationale (see sect. 2.2), students’ answers to questions Q3–Q5 have been labeled as either logically correct, if the relations between problem instances are correct, or appropriate for recursion, if, moreover, the given procedure is applied to “simpler” arguments. Then the data have been aggregated by counting, for each student, the number of (at least) logically consistent answers and the number of correct answers that are also appropriate for recursion.
The answers to items Q6 and Q7 have been classified as either structurally recursive or simply consistent (refer again to sect. 2.2). A variety of consistent, but not structurally recursive solutions may result from an attempt to adapt an iterative approach, familiar to most students. This seems to be the case of the programs provided in Q7 that achieve the goal by scanning in the forward direction the argument string to count the number of parentheses of either type, or to match the string with a reversed copy of it.

As observed in section 2.2, the syntactically correct assemblies of item Q8 may represent either well-founded or ill-founded definitions. In the case of Q9, definitely the easiest question, we can discriminate strict and loosen adherence to Scheme syntax (e.g., in the use of parentheses).

The answers to the last two items have been classified on the basis of the emergent level of abstraction: high-level if a precise functional characterization is provided for at least one (of the two) recursive procedure in Q10, or if appropriate (sub)procedures are introduced to re-factor the program in Q11; moderate-level if only the trivial procedures in Q10 are properly characterized, or if only let expressions are used in Q11. The procedure names suggested in Q10 were only considered to resolve possible ambiguities of the characterizations.

As a final methodological remark, the analysis of all the items in part I has taken into account the answers to the individual questions proposed in the second session.

### 4.3 Results

I will start this section by considering the outcome relative to the computation model. Fig. 1 illustrates the results of the analysis of the traces of recursive computations (Q1, Q2) and compares them with the findings of [16] (lowest bar).

Then, to have a grasp of the whole picture, we can relate the feedback for each learning dimension to some degree of fluency with the computation model. To simplify matters, I will distinguish three levels of understanding of the operational aspects: sound (58% of the students in 2009, 69% in 2008), if the learners apply consistently a sound model to Q1 and a sound or fragile model to Q2; uneven or unclear (16% in 2009, 4% in 2008), if they apply a sound model either to Q1 or Q2, but not to both; unsound or not developed, if the students are not able to provide correct traces.

The arrangement of the bars in fig. 2 exploits these three categories, that partition the upper bar, to correlate the different kinds of answers to Q3–Q7 with the solidity of the computation models. The bars left-aligned in each column refer to the same group of learners. The data are relative to the a.y. '09-10 and the horizontal lengths represent numbers of students (50 in total). Similarly, fig. 3 correlates the answers to Q8–Q11 with the soundness of the computation models, but for a smaller sample (38 students).

#### Relations in the problem domain.

The second and third rows in fig. 2 report the numbers (3, 2, or 1, visualized in different colors) of logically correct, respectively appropriate for recursion, answers to items Q3–Q5. A more detailed account concerning this learning dimension is given in tab. 1, where the figures in the second column are inclusive of the answers appropriate for recursion.

#### Recursive structures.

The bars in the last two rows of fig. 2 represent in different colors structurally recursive, consistent and useless (paler tone) answers to Q6 and Q7. The percentages of positive answers are also reported in tab. 2, where a structurally recursive characterization counts also as a consistent characterization.

#### Language structures.

87% of the solutions of Q8 are syntactically correct, but only 34% also define a well-founded recursive procedure. Relative to Q9, the adherence to Scheme syntax is strict in 84% of the cases, loose in the 8%. These data are also visualized in the second and third rows of fig. 3, where the paler tone means incorrect syntax.

#### Abstraction.

The colors in the last two rows of fig. 3 discriminate between high, moderate and low (pale tone) levels of abstraction. The main figures on the abstraction ability (items Q10 and Q11) are also summarized in tab. 2.

<table>
<thead>
<tr>
<th>Year</th>
<th>2009</th>
<th>2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>recursion:</td>
<td>Q6</td>
<td>Q6</td>
</tr>
<tr>
<td>consistent</td>
<td>42%</td>
<td>40%</td>
</tr>
<tr>
<td>structural</td>
<td>10%</td>
<td>11%</td>
</tr>
<tr>
<td>abstraction:</td>
<td>Q10</td>
<td>Q10</td>
</tr>
<tr>
<td>high-level</td>
<td>21%</td>
<td>21%</td>
</tr>
<tr>
<td>moderate-level</td>
<td>39%</td>
<td>39%</td>
</tr>
</tbody>
</table>

Table 1: Students’ answers to items Q3–Q5.

<table>
<thead>
<tr>
<th>Year</th>
<th>2009</th>
<th>2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 answers</td>
<td>32%</td>
<td>16%</td>
</tr>
<tr>
<td>2 answers</td>
<td>20%</td>
<td>27%</td>
</tr>
<tr>
<td>1 answer</td>
<td>16%</td>
<td>33%</td>
</tr>
</tbody>
</table>

Table 2: Students’ answers to Q6, Q7, Q10, Q11.
correlation is moderate (≈0.5) between dimensions (iii) and (v). This may mean that, to some extent, the corresponding abilities are developed independently of each other.

6. CONCLUSIONS

I have summarized the results of an investigation to address several dimensions of learning recursion. After a replication of the experiment, I am reasonably convinced of the consistency of the findings. In my view, two main conclusions can be drawn from the study: (i) neither the language syntax nor the computation model are crucial for the learning process; (ii) we should spent more effort on the declarative aspects implied by programming, since “The key to comprehending any form of abstraction, including recursion, is to focus on the what and down play the how” [17]. As a possible direction of future work, the answers to the questionnaire could be correlated to the students’ performances in the exams. Moreover, it will be interesting to design a questionnaire with a similar structure to address iteration.

7. REFERENCES