Platyhelminthes Are [Re]constructed Recursively*

Alberto de la Encina[1], Mercedes Hidalgo-Herrero[2], and Olga Marroquín-Alonso[1]

[1] Dpto. Sistemas Informáticos y Programación, Facultad de Informática
[2] Dpto. Didáctica de las Matemáticas, Facultad de Educación
Universidad Complutense de Madrid, Spain
albertoe@sip.ucm.es, mhidalgo@edu.ucm.es, alonso@sip.ucm.es

Abstract. In this paper we propose a progressive technique for teaching recursion with the aim of making easier the understanding of this concept by students of other areas than computer science. Since knowledge is intended to be actively constructed by the students and not passively absorbed from textbooks and lecturers, the adopted teaching technique is derived from constructivism. This paper presents a study of the results obtained in 2005 by two groups of students from Mathematics and Statistics degrees.

1 Introduction

During the last years, computational science has been proved to be a powerful tool in many areas such as economy, mathematics, statistics, biology, and so on. Therefore, the corresponding curricula contain basic courses on computer science, since the students involved in these areas should learn its basic concepts. This applied aspect of computer science has led us to study its methodology in Mathematics and Statistics degrees, in order to check the understanding level of its foundations. In our opinion, recursion should be included in such degrees, owing to its suitability in program design [10]. In fact, recursive-functional languages have proven their usefulness in simplifying programming efforts when dealing with complex problems in computational science (see e.g. [6]). In this respect, there exist two generalized tendencies in teaching recursion: In a basic course or as an advanced tool [1]. We consider that it should be taught as soon as possible, so that students can both take profit from it during their studies, and by doing so, strengthen their knowledge about this useful technique. In this paper, we have taken both approaches: Our first choice was considering recursion as a basic tool, but its treatment as an advanced one is also present in our experiment, because the individuals in the group of Mathematics hardly received training in recursion during their first years.

In addition to this, we pretended to introduce recursion as a natural tool for solving problems whose iterative solution is highly complex. Students were previously taught *if* statements, loops, and subprograms. The new approach was conceived by considering not only our teaching experience, but also the unfruitful traditional methodology [11] and the initial iterative tendency of students.

* Work partially supported by the Spanish MCYT project TIC2003–07848–C02–01.

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part II, LNCS 3992, pp. 207–214, 2006.

1.1 Didactical Framework: Constructivism

The traditional methodology for teaching consists in transmitting knowledge by means of teacher explanations. By contrast, we will consider *constructivism* [2], where the teacher task is to pose problems for being solved by the students. In such a case, the teacher just observes the strategies employed by the scholars, but without revealing the solutions of the proposed questions [5].

Besides, thought, as an essential mechanism for solving logical/mathematical questions, comes from acting over an environment [9]. Moreover, the experimentation must be both constant and organized [4], and as a consequence, the teacher will guide the learning of the students by organizing the posed problems. The first one must be easily solvable by modifying the previous knowledge, and the remaining sequence will be proposed by introducing *didactic variables*, that is, planned items which can be modified by the teacher in order to sway the students hierarchy of strategies for solving problems [3, 7].

Following this constructivist methodology, we have used questions dealing with platyhelminthes, that is, flatworms with two kinds of reproduction which allow us to recursively study their family tree. Our proposal comprises four sections: "General definition", "base case/recursive case", "final recursion/non-final recursion", and "inefficiency of several recursive calls".

2 Our Proposal

This section comprises the problems posed to the students by paying special attention to the used didactic variables and their purposes. The solving of such problems was performed in the following way:

1. Students were given the formulations.
2. Problems were solved by the individuals by following the given instructions. The teacher behaved as constructivism states.
3. Students explained their solutions. Afterwards, the most suitable ones were selected. In this framework, the discussion among students is extremely important so that the validation of the results is carried out by themselves.

The use of a programming language was not compulsory, so specifications in natural language were also admitted. The problems were the following: Let us consider a colony of platyhelminthes which is characterized by both its way of reproduction (every worm has at least one children) and the immortality of its individuals. Answer concisely the proposed questions. The following functions[1] can be used:

1. `fun hermaphrodite(w:worm) returns (b:bool)` where `hermaphrodite(w)` $\leftrightarrow$ `w` is an hermaphrodite worm (sexual reproduction)[2].
2. `fun parent(w:worm) returns` (w_r`:worm`) where w_r either generated `w` by fragmentation (asexual reproduction) or is the male worm which generated `w` by internal fecundation (sexual reproduction).

[1] They are written by using the notation defined in [8].
[2] The way of reproduction of a worm is constant in its lifetime.

3. `fun children(w:worm) returns (ch_s:v)` where `type v=array [1..n] of worm` and $\texttt{ch}_\texttt{s}$ contains the children of worm `w`.
4. `fun firstborn_child(w:worm) returns (w_r:worm)` where $\texttt{w}_\texttt{r}$ is the firstborn child of `w` (generated from either `w`'s head or its first fertilized egg).
5. `fun name(w:worm) returns (s:string)` where `s` is the first name of `w`.
6. `action write(s:string)` writes the string `s`.

2.1 General Definition

The first problem deals with the concept of *recursion* by means of two questions which ask for the ancestors and the descendants of a given worm, respectively.

In order to avoid several ascending branches in the family tree, the didactic variable *way of reproduction* takes the value *by fragmentation*. In the second question, fragmentation also sets the number of descendants at two.

1. Let us consider that there did not exist an original worm in the colony such that the remaining worms come from it. Define the set of ancestors of a given flatworm `w` if all the reproductions leading to it have been by fragmentation. Similarly, define the set of descendants of platyhelminth `w` if its reproduction and those of its descendants have been by fragmentation.

Because of the way of reproduction of the ancestors, the solution for the first question is a *linear* or *simple recursive function*, whereas the second one is solved by using a *non-linear* or *multiple recursive function*.

Notice that the problem asks for neither the cardinal of ancestors nor the one of descendants of `w`. Defining a function to calculate the number of ancestors is trivial, whereas the number of descendants can be expressed by the following geometric progression: $\sum_{n=1}^{\infty} 2^n$.

Anchor cases are not possible in solving these questions because the colony features make all the branches in the family tree infinite.

2.2 Base Case/Recursive Case

In order to introduce the *base* or *anchor case* concept, we modify the previous formulation by combining both values for the variable *way of reproduction*.

2. Let us consider a non-hermaphrodite flatworm `w`. Define a function to work out the number of ancestors of `w` whose reproduction is by fragmentation. The calculation must finish when getting the youngest ancestor whose reproduction has been sexual. Similarly, define a function to calculate the number of descendants of `w` which have bred asexually and whose parent and sibling are asexual as well.

The first function base case is satisfied by any ascending branch where a sexual worm exists. Similarly, the second function anchor case is met by a descending branch as long as there exists an asexual worm with an hermaphrodite child. The calculation of descendants requires two recursive calls and a base case, whose definition is simplified by the imposed condition on the descendants of `w`.

2.3 Final Recursion/Non-final Recursion

In order to solve the following problem a *non-final recursive function* is needed. Even though the previous questions required non-final recursion, now our purpose

goes further: The order between the result processing and recursive calls becomes crucial when defining the required sets. This significance is due to the introduction of the didactic variable *age order* with two possible values: *From the eldest to the youngest* and *from the youngest to the eldest*. Besides, the variable *way of reproduction* allows to obtain both a simple sequence of platyhelminthes (ancestors of `w`) or a graph/tree of flatworms (descendants of `w`).

3a. Let us assume that all the flatworms in the colony have reproduced by fragmentation. In these conditions, the firstborn child of a platyhelminth is the one generated from its head. Besides, let us consider that all the flatworms have a first name which identifies them univocally. Given a platyhelminth `w`, enumerate its firstborn descendants from the eldest one to the youngest one.

3b. Let us suppose a combination of reproduction types where every flatworm has at least one hermaphrodite descendant. In these conditions, the firstborn child of a platyhelminth is the one either generated from its head (asexual reproduction) or born from its first fertilized egg (sexual reproduction). Besides, let us suppose that all the flatworms have a first name which identifies them univocally. Given a non-hermaphrodite platyhelminth `w`, enumerate its firstborn non-hermaphrodite descendants. The calculation must finish when getting the first sexual one and the enumeration must be from the eldest one to the youngest one.

3c. Let us assume the conditions in 3b. Given a non-hermaphrodite platyhelminth `w`, enumerate the first names of its non-hermaphrodite firstborn descendants. The calculation must finish when a sexual worm is reached and the enumeration must be from the youngest one to the eldest one.

It is easy to see that questions 3a and 3b[3] are *final recursion* cases, whereas the solution of question 3c is a non-final recursive algorithm.

2.4 Inefficiency of Several Recursive Calls

Recursion drawbacks encompass inefficiency because the usual overload produced by algorithm invocations is increased by using this mechanism. The reason is that an external call may generate a lot of internal invocations.

The following problem will make students face this drawback as they are asked to design a method for improving recursion efficiency.

4a. Let us consider that in the colony there existed an original flatworm which is the eldest ancestor of all the platyhelminthes there. Besides, each worm is bigger than its ancestors as shown by the following expression: `rings(w)=rings((parent(w))+(rings(parent(parent(w)))div2)`. Calculate the number of times each platyhelminth number of rings is computed when the final task is calculating the number of rings of a great-great-grandchild of the original flatworm[4].

By analyzing the simplest solution of this question it is clear that some recursive algorithms are inefficient owing to the used design method. In such a case, the same values are calculated many times, though these repeated computations can be avoided by using auxiliary parameters.

[3] In question 3b, the writing action cannot be included in the base case.

[4] The students were given the following data: `rings(firstborn_child(w0))=4` and `rings(w0)=2` where `w0` is the original flatworm.

4b. Let us assume the conditions in question 4a. Calculate the number of rings of a flatworm in such a way that each ancestor is processed only once.

3 Obtained Results: Mathematics

The detailed analysis of the solutions proposed by the students of the subject *Java Programming* allowed us to check the fact that they hardly know the concept of recursion except from the idea that a recursive function or procedure calls itself repeatedly. As a consequence, many students answer with iterative algorithms to inherently recursive questions, such as the enumeration of both the ancestors and descendants of a given flatworm. This iterative tendency yields never-ending loops whenever the recursion we are dealing with lacks a base case. Next, we discuss such design issues by analyzing some of the obtained results.

3.1 General Definition

More than half of the students (seven of 13) defined iterative algorithms based on loops instead of recursive functions. A representative fragment of (Java) code extracted from these algorithms is the following, where the boolean condition is trivially satisfied because there did not exist an original worm in the colony.

```
worm wr=w; while(parent(wr)!=null){wr=parent(wr);}
```

In addition to this, just one of the six remaining students deemed the question too difficult, but the rest of solutions were correctly formulated in a recursive way. Next, we show the definition of the set of ancestors of `w` provided by the only student who learnt Haskell in his first year of Computer Science. The analysis of his answers confirms the idea that teaching functional programming languages before imperative ones makes easy the understanding of the concept of recursion:

$\mathtt{y} \in \mathtt{A}(\mathtt{w}) \leftrightarrow \mathtt{y} = \mathtt{parent}(\mathtt{w}) \vee \mathtt{y} \in \mathtt{A}(\mathtt{parent}(\mathtt{w}))$

3.2 Base Case/Recursive Case

In this case, nine students defined a function to work out the number of ancestors of `w` and seven provided a function to calculate the number of descendants of the same worm. In spite of this low level of response, the proposed algorithms such as the following show a clear iterative tendency in defining recursive concepts.

```
worm wr=w; boolean asexual=true;
while (asexual){
 if (hermaphrodite(parent(wr))) asexual=false else wr=parent(wr);}
```

Seven (of nine) students provided similar algorithms to the one shown above, whereas there was just one suitable solution recursively defined. Concerning the calculation of the descendants of `w`, many functions were not properly defined owing to the fact that they were based on the expression

$\mathtt{hermaphrodite}(\mathtt{parent}(\mathtt{w})) \vee \mathtt{hermaphrodite}(\mathtt{sibling}(\mathtt{w}))$, where

$\mathtt{sibling}(\mathtt{w}) = \mathtt{w_r} \leftrightarrow \mathtt{w_r} \in \mathtt{children}(\mathtt{parent}(\mathtt{w})) \wedge \mathtt{name}(\mathtt{w_r}) \neq \mathtt{name}(\mathtt{w})$,

while a closer analysis of the question yields the boolean condition

$\mathtt{hermaphrodite}(\mathtt{children}(\mathtt{g})[1]) \vee \mathtt{hermaphrodite}(\mathtt{children}(\mathtt{g})[2])$.

Therefore, two (of seven) students provided an iterative algorithm by using the first expression, and five of them defined a recursive implementation, two of which were correctly designed. Finally, we note that students tend either to solve both questions (seven of 13) or to leave blank both of them (four of 13). The two remaining students provided only an algorithm to work out the number of ancestors of `w`.

3.3 Final Recursion/Non-final Recursion

As expected, the number of students who tried to solve a problem is inversely proportional to the level of difficulty of such problem. In this way, seven students provided solutions corresponding to questions 3a and 3b, whereas question 3c was solved by four students. In any case, loops were less used than before, though it was easy to find in them many mistakes quite similar to those previously described. The proposed algorithms dealing with final recursion were correct in most cases, as shown by the following functional definition:

```
names x = name x : names firstborn_child x
```

Nevertheless, non-final recursion proved to be hard to understand, since in order to solve the last question, students defined a double path through the family tree: First the youngest sexual descendant of `w` is found, and then its ancestors are properly displayed. In short, four students solved properly questions 3a and 3b, one of whom provided a correct algorithm in Haskell for question 3c.

3.4 Inefficiency of Several Recursive Calls

The question was solved by only three students, who also provided algorithms for the previous problems. They correctly calculated the number of rings of a great-great-grandchild of the original worm in the colony, but the provided solutions for computing such number in the general case were not accurate. To be exact, just one student proposed a correct solution in Haskell for the whole problem, whereas his schoolmates, who learnt Pascal in their first year of Computer Science, did not handle accumulators properly.

4 Obtained Results: Statistics

The results in this section show the difficulties in teaching recursion owing to the iterative tendency of human mind. As a consequence, our main purpose is to make students understand recursion as a natural mechanism for problem solving.

Besides, these individuals, who study statistics, are not familiarized with programming techniques, so they are not supposed to give formal solutions for the posed questions but informal specifications of the corresponding algorithms.

4.1 General Definition

The first approach for the first question (ancestors of `w`) given by the 14 students was iterative, so that the following hint was given: "Let us assume that one of your ancestors knows the set of his/hers. Could you answer this question by

using such set?". This clue led the students to a correct recursive algorithm, though it was fairly suggested by the teacher. Concerning the question about the descendants of w, one student provided an iterative algorithm, whereas the remaining 13 proposed a solution defined in a recursive way, one of which was correct. The rest of recursive algorithms called themselves just once, maybe because in those days statistics students had not learnt arrays in depth.

4.2 Base Case/Recursive Case

The number of clues given by the teacher diminished considerably for this problem, owing to the fact that recursion was introduced in the previous one. In spite of this, not only three students provided iterative algorithms, but also another two defined solutions by combining loops with recursive calls in the non-trivial case. The following fragment of code shows such combination.

```
repeat anc:=parent(w)+ances(parent(w)) (*recursive call*)
until (hermaphrodite(parent(w))=true)
```

Nine students proposed recursive solutions, two of which lacked a base case.

4.3 Final Recursion/Non-final Recursion

According to the constructivist method, the teacher did not gave any hint in this case, since it was assumed that recursion was introduced enough. In some cases, the traverse of the family tree was wrongly accomplished, due to the fact that just two generations were reached, as shown by the following implementation.

```
nameW:=name(w)+name(firstborn_child(w))
```

However, other recursive solutions were quite clear. We show the most representative one corresponding to question 3b.

```
fun list(w:worm) returns (n:string)
 n=name(w)+if (hermaphrodite(firstborn_child(w)))
                list(firstborn_child(w))
```

Finally, the analysis of the obtained results confirms that students tend to consider recursion when solving those problems whose iterative solutions have an increasing complexity. The ratio between the number of iterative solutions and the total number of answers were: Ten to 14 in question 3a, five to ten in question 3b, and one to eight in question 3c. Besides, we noticed that half of the recursive algorithms provided for questions 3b and 3c did not consider a base case, though these solutions were more correct than the iterative ones.

4.4 Inefficiency of Several Recursive Calls

As expected, only few students tried to solve this problem, as it shows the fact that only six individuals provided a solution for the first question. Besides, every student except one mistaken the calculations in the following way: Given the expressions

$\texttt{rings}(w_4)=\texttt{rings}(w_3)+(\texttt{rings}(w_2)\texttt{div}2)$,
$\texttt{rings}(w_3)=\texttt{rings}(w_2)+(\texttt{rings}(w_1)\texttt{div}2)$, and
$\texttt{rings}(w_2)=\texttt{rings}(w_1)+(\texttt{rings}(w_o)\texttt{div}2)$

they did not consider that platyhelminth w_2 is consulted three times, owing to the number of rings of w_2 is used for processing w_4 and w_3. This mistake is due to the fact that students did not develop the whole invocations tree, that is, they just worked out for each flatworm its appearance in the right hand-side of the equations. Finally, the last question was answered by four students, who failed to solve the involved recurrence and did not propose any recursive solution.

5 Conclusions and Current Work

The main purpose of this test was the study of teaching recursion in other areas than computer science. It can be observed that recursion can be successfully taught in a basic course in order to provide a useful technique for developing computational systems by students of either Mathematics or Statistics degrees. Concerning our platyhelminth problem, it is easy to see that students tend to provide iterative algorithms to solve the simplest questions, whereas recursion is used in the complex ones, because recursive programming needs fewer low level details than loop programming. In this way, the teacher's role as a simple guide has proved to be very useful, since students have evolved by themselves from iterative to recursive solutions. Besides, since the posed problems deal with family trees (a familiar concept) instead of the typical mathematical items (e.g. the Fibonacci series), its use in areas such as biology or economy is more plausible.

As current work, we pretend to pose again these questions by simplifying their formulations with the purpose of making them suitable for other programming subjects and areas.

References

1. O. Astrachan. Self-reference is an illustrative essential. In *Proceedings of SIGCSE 1994, Technical Symposium on Computer Science Education*, pages 238–242. ACM Press, 1994.
2. M. Ben-Ari. Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, 20(1):45–73, 2001.
3. J. Briand and M. C. Chevalier. *Les Enjeux Didactiques dans l'Enseignement des Mathématiques*. Hatier, 1995.
4. J.S. Bruner. *On Knowing: Essays for the Left Hand*. MA:Harvard University Press, 1962.
5. M. Chamorro, J. M. Belmonte Gómez, S. Llinares, M. Ruiz Higueras, and F. Vecino-Rubio. *Didáctica de las Matemáticas para Primaria*. Pearson, 2003.
6. A. Encina, I. Rodríguez, and F. Rubio. Testing spculative work in a lazy/eager parallel functional language. In *Languages and Compilers for Parallel Computing (LCPC'05)*, LNCS. Springer-Verlag, 2006. In press.
7. M. Hidalgo-Herrero, I. Rodríguez, and F. Rubio. Testing learning strategies. In *Int. Conference on Cognitive Informatics (ICCI'05)*, pages 212–221. IEEE, 2005.
8. R. Peña. *Diseño de Programas. Formalismo y Abstracción*. Prentice Hall, 1998.
9. J. Piaget. *Introduction à l'Épistemologie Genetique*. PUF, 1973.
10. A. B. Tucker. Computing curricula 1991. *Communication of the ACM*, 34(6):68–84, 1991.
11. J. Velázquez-Iturbide. Recursion in gradual steps (is recursion really that difficult?). In *Proceedings of SIGCSE 2000, Technical Symposium on Computer Science Education*, pages 310–314. ACM Press, 2000.