Chapter 15
Interactive Learning of Recursion

Antonio Pérez-Carrasco
Universidad Rey Juan Carlos, Spain

J. Ángel Velázquez-Iturbide
Universidad Rey Juan Carlos, Spain

ABSTRACT
One concept that has proved to be especially difficult to comprehend in computer science education is recursion. This chapter provides an overview of past efforts on the teaching of recursion. The authors first introduce concepts and models about the teaching and learning of recursion. In particular, they identify models used by teachers to explain recursion (i.e. conceptual models) and models used by students in their learning process (i.e. mental models). Afterwards, they review the teaching methods used in the past. Finally, the authors survey visualization and animation systems for recursion, explaining how they support conceptual models and how they try to remove wrong mental models. They also include a comprehensive technical comparison of the systems and review the evaluations these systems have been subject to.

INTRODUCTION
Programming education has been for about forty years a challenge for computer science educators. Not surprisingly, many pedagogues and computer scientists devoted huge efforts to enhance its education (Robins, et al., 2003). Many programming issues have been an area of educational research in past years, namely students understanding, teaching methods, educational technology, visualization and animation, assessment, etc. One concept that has proved to be especially difficult to comprehend is recursion, but there is consensus on considering it necessary for computer science students (Astrachan, 1994).

The chapter presents past efforts aimed at interactive learning of recursion. We first introduce concepts and models about the teaching and learn-
Interactive Learning of Recursion

ing of recursion. In the third section, we review the range of teaching methods proposed in these years. Finally, we survey existing systems for recursion visualization and animation, making emphasis on the support each system gives to educational issues and models. We also include a comprehensive technical comparison of the systems and review the evaluations these systems have been subject to.

MODELS OF RECURSION

Recursion is a fundamental, advanced construct for program design. In this section we introduce recursion on a three-folded basis. Firstly, we briefly define recursion and identify its main elements. Secondly, we show alternative ways of explaining recursion by an instructor. Finally, we present the most common students’ errors on understanding recursion.

Recursion

A recursive definition defines something in terms of itself. This property is strongly discouraged for definitions in general, and dictionaries try to avoid it. Recursion can be direct but also indirect, being this form of recursion harder to recognize. For instance, the definition of a word in a dictionary depends on other words, and this dependence chain is potentially infinite. It is not rare that, following a chain of dependencies, the definition of the original word depends on itself. However, in mathematics and computer programming, recursion is a valuable tool for definitions and for problem solving. Some recursive definitions are surprisingly clear. For instance, the factorial of a natural number \( n \) can be symbolized as \( n! \), meaning that \( n! = n \cdot (n-1) \cdot \ldots \cdot 1 \). An alternative, recursive definition follows:

\[
 n! = \begin{cases} 
 1 & \text{for } n = 0 \\
 n \cdot (n-1)! & \text{for } n > 0 
\end{cases}
\]

The term at the left-hand side is called the definition head. A recursive definition consists of two or more cases, at its right-hand side. A case for which the concept being defined does not depend on itself is called a base case. A case which depends on the concept being defined is called a recursive case; the recursive occurrence of the term defined is called a recursive call.

Many additional, typical examples exist. For instance the Fibonacci numbers are typically defined as follows:

\[
 \begin{align*}
\text{fib}(n) &= \begin{cases} 
 1 & \text{for } n = 0 \\
 1 & \text{for } n = 1 \\
 \text{fib}(n-2) + \text{fib}(n-1) & \text{for } n > 1 
\end{cases} \\
\end{align*}
\]

As in the dictionary example, recursion can be indirect. The following is an alternative, mutually recursive definition of the Fibonacci numbers:

\[
\begin{align*}
\text{fib}(n) &= a(n) + b(n) \\
a(n) &= \begin{cases} 
 0 & \text{for } n = 0 \\
 a(n-1) + b(n-1) & \text{for } n > 0 
\end{cases} \\
b(n) &= \begin{cases} 
 1 & \text{for } n = 0 \\
 a(n-1) & \text{for } n > 0 
\end{cases}
\end{align*}
\]

There are many particular cases of recursion with different features: lineal vs. multiple recursion, final vs. non-final (lineal) recursion, direct vs. mutual recursion, etc. We do not deepen into the definition and explanation of recursion, as it is common in programming or mathematical logic textbooks. However, we summarize the constraints a well-formed recursive definition must respect:

- It must contain at least one base case and one recursive case.