SRec: An Animation System of Recursion for Algorithm Courses

J. Ángel Velázquez-Iturbide, Antonio Pérez-Carrasco, Jaime Urquiza-Fuentes
Departamento de Lenguajes y Sistemas Informáticos I
Escuela Superior de Ingeniería Informática
Universidad Rey Juan Carlos
C/ Tulipán s/n, 28933 Móstoles, Madrid, Spain
{angel.velazquez,antonio.perez.carrasco,jaime.urquiza}@urjc.es

ABSTRACT

In this paper we describe SRec, a system to animate recursion in Java programs. It is intended to assist in algorithm courses to better understand and analyze algorithm behavior. We make several contributions. Firstly, SRec exhibits a comprehensive set of animation and educational features. It provides three complementary, coordinated views of recursion: traces, the execution stack and activation trees. SRec allows the user constructing and modifying animations without effort. The animation can be played flexibly, both forward and backwards. It also provides facilities to integrate animations into courses. Secondly, the paper describes the educational features of the system and its use in algorithm courses. Thirdly, the system has been fully evaluated with respect to usability (using formative and summative methods) and has been compared to other systems reported in the literature. The results of both evaluations are highly positive.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features – *recursion*; K.3.1 [**Computers and Education**]: Computer Uses in Education; K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education*.

General Terms

Algorithms, Experimentation, Human Factors, Languages.

Keywords

Program animation, recursion, activation trees, usability.

1. INTRODUCTION

Recursion plays an important role in algorithm courses, where it is not a concept to learn but a concept to apply. Recursion is the basis of divide-and-conquer algorithms. Dynamic programming algorithms are usually stated recursively in a first step; afterwards, they are rewritten as tabulated, iterative algorithms. Greedy algorithms are stated recursively and implemented iteratively. Backtracking algorithms can be coded in different ways, typically combining iteration and recursion. Consequently, a system to animate recursion may be a valuable tool for an algorithm course.

In this paper we introduce SRec, a system to animate recursion. We can find in the literature a number of graphical representations of recursion; some of them are general and widely accepted [7], whereas others are more particular [11]. SRec displays three well-known representations: traces, the execution stack and activation trees. Animations can be played both forward and backwards.

The structure of the paper is as follows. In the second section the main functions of SRec are described. The third section shows its educational features and illustrates its use in algorithm courses. The fourth and fifth sections contain the results of a usability evaluation and of a comparison with respect to related systems. Finally, we state our main conclusions and future work.

2. DESCRIPTION OF SRec

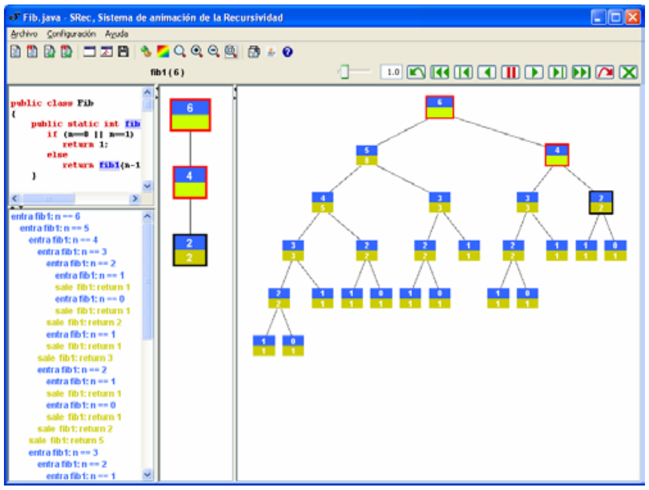
In this section, we explain the main features of SRec. Firstly, we give an overview of its visualization and animation facilities. We then describe basic and advanced elements of its visualizations.

2.1 Visualization and Animation

Fig. 1 shows a screen snapshot of SRec obtained while computing the 6th Fibonacci number. The screen is divided into four panels. The upper left panel contains the source code in Java. The other three panels contain three graphical representations, namely, left to right: trace, execution stack, and activation tree. The animation controls are at the upper right side.

SRec was built using an effortless approach [5] to animation based on the automatic generation of visualizations related to source code (program visualizations). When an algorithm is executed, a visualization of each successive state is generated as a side effect. The graphical representation of a state depends on the particular paradigm or domain. An animation consists in playing the generated visualizations by means of a flexible set of controls.

In SRec, the user can generate an animation from a Java source file by following several steps. Firstly, the user preprocesses the Java file by selecting it and marking one or more methods as "visualizable". Secondly, the user selects the main method and arguments of the initial call of the animation. Argument values may be typed with the keyboard, loaded from a file, or generated in any valid way (e.g. the contents of arrays may be generated in ascending, descending or random orders). Thirdly, a main call is executed and a trail of events is generated, formed by call and return statements executed by visualizable methods. Fourthly, SRec displays the source file in the code panel, sets the three visualization panels to the start state, and gets ready to play the animation. Finally, the user can play a discrete animation of the visualizations corresponding to the trail of events.



```
public class Fib
{
    public static int fib
        if (n==0 || n==1)
            return 1;
        else
            return fib1(n-1
    }
```

```
entra fib1: n == 6
  entra fib1: n == 5
    entra fib1: n == 4
      entra fib1: n == 3
        entra fib1: n == 2
          entra fib1: n == 1
          sale  fib1: return 1
          entra fib1: n == 0
          sale  fib1: return 1
        sale  fib1: return 2
        entra fib1: n == 1
        sale  fib1: return 1
      sale  fib1: return 3
      entra fib1: n == 2
        entra fib1: n == 1
        sale  fib1: return 1
        entra fib1: n == 0
        sale  fib1: return 1
      sale  fib1: return 2
    sale  fib1: return 5
    entra fib1: n == 3
      entra fib1: n == 2
        entra fib1: n == 1
```

Fig. 1. A screen snapshot of SRec

The animation controls are similar to those in a VCR set (see Fig. 2). In addition to the pause and exit actions, the following controls are available (at the right of the pause button): continuous play, advance one step, go to the end, and jump over the current recursive call. The same operations are available in backwards direction, and have the reverse semantics. Speed of continuous animation can be set either with a tuning bar or by introducing a number.

As the user interacts with the control bar, the three visualization panels are updated simultaneous and consistently. Some control buttons are sometimes forbidden, being this state highlighted in red. For instance, the pause action is not allowed in Fig. 2 because the user is not playing the animation continuously.

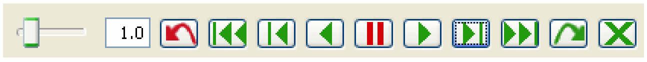


Fig. 2. Control bar of animation

2.2 Basic Graphical Representations

SRec provides three graphical representations of recursion. It coherently updates the three visualization panels every time a new call is entered or exited. In the first case, the values of the arguments are shown; in the second one, the result is displayed.

An activation tree [7] is a hierarchical representation that shows recursive invocations. Each node is associated to a recursive call and is composed of two areas: the top area with the arguments values and the bottom with the result. (Notice that a related, more familiar representation is the recursion tree, where each node only contains the value of the arguments.)

A second representation is the execution stack (or control stack). It is implementation-oriented, and contains the activation records of pending calls. SRec displays the same amount of information within nodes of either the stack or an activation tree. For simplicity, other data present in an actual implementation are omitted, such as the return address or local variables.

The third representation is a trace. It is a common view in functional languages that provides a temporal description of the recursive process, consisting in printing an indented line per each entry or exit of an invocation.

The animation buttons allow playing an animation. Most of them have an intuitive semantics. We describe here in detail the semantics of the two most complex actions: advance one step and jump over a recursive call.

The action of advancing one step is equivalent to the *step into* action of debuggers. Its effect over an activation tree depends on the current state. We can distinguish four cases (see Fig. 3):

- When the execution is at the entry of a recursive case, the first recursive call that it invokes is displayed with its arguments (Fig. 3.a).
- When it is at the entry of a base case, its result is displayed (Fig. 3.b).
- When it is at the exit of a call and the call that invoked it has pending calls, the next (sibling) recursive call invoked is displayed with its corresponding arguments (Fig. 3.c).
- When it is at the exit of a call and the call that invoked it has no pending calls, the result of the latter is displayed (Fig. 3.d).

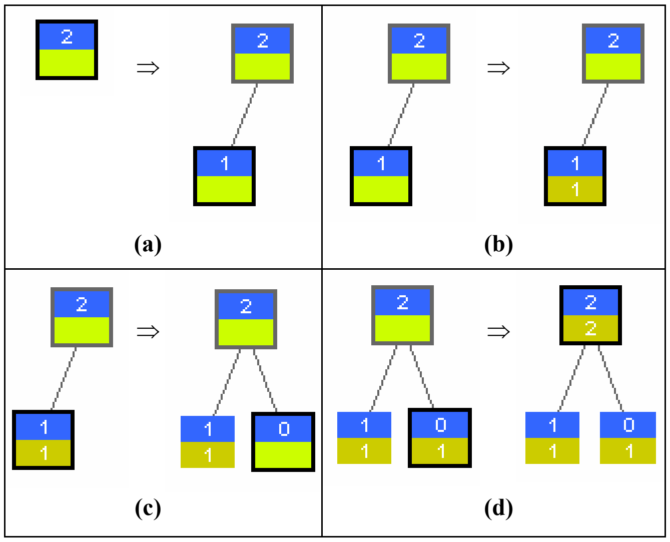


Fig. 3. Effect over an activation tree of advancing one step

The action of jumping over the current recursive call is equivalent to the *step over* of debuggers. This action can only be executed when it is at the entry of a recursive call, i.e. one of the two former situations described above. As a consequence, the result of the call and the (recursively) generated subtree are displayed.

A useful feature of SRec consists in highlighting the active call with a frame (in the previous figures, in black). Thus, the execution focus is visually remarked to locate it easily. Analogously, the path that links the tree root to the active call has its nodes framed in another color (above, in red or grey).

2.3 Advanced Graphical Features

SRec provides some facilities to handle large visualizations. The four panels can be adjusted to the sizes of the visualizations, or even collapsed. SRec also provides a pan+zoom interface [2] (i.e. scrolling and zoom) to navigate through large visualizations. Several zoom functions are available for the views of activation trees and execution stack (enlarge, reduce, perfect adjust).

SRec also offers a large range of customization options for the three graphical representations. Customization is performed over a large set of features: colors, fonts, sizes, lines and distances.

More importantly, SRec allows controlling the amount of data displayed in activation trees. Firstly, each node may contain input data, output data, or both simultaneously; this option propagates to the execution stack view. Secondly, the *jump over* action may display in detail the subtree generated or just its result. Thirdly, three variants of activation trees are supported (see Fig. 4):

- Activation trees with history. The information of past calls is kept visible.
- Activation trees with attenuated history. The information of past calls is visible, but in an attenuated color.
- Activation trees without history.

The three graphical representations of SRec are closely related. The execution stack contains the same nodes as the path of the activation tree that links the root and the active call. The trace is equal to the union of the preorder traversal of input values in the activation tree and the postorder traversal of output values.

These relations are reinforced by SRec in several ways. Firstly, SRec updates the three views consistently after each animation action. Secondly, values of the same kind are displayed in the same colors for the three representations. Thirdly, the equality of the execution stack and the active path is reinforced by stacking the calls top down in both representations and by framing them with the same colors (see Fig. 1).

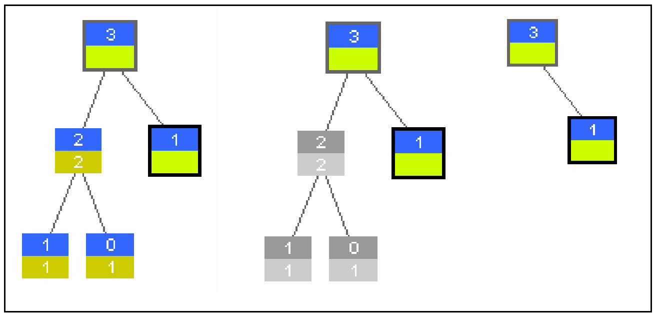


Fig. 4. Three variants of the same activation tree

3. EDUCATIONAL FACILITIES

SRec is intended for educational use. We explain here its most relevant educational features and its use in algorithm courses.

3.1 Educational Features

SRec is generic for any (recursive) algorithm and for any input/output primitive data type. It also provides configuration options and immediate update of visualizations. As a consequence, SRec fosters an exploratory style of interaction.

The automatic generation of animations and the availability of atomic operations to store/load animations reduce the workload of the user of SRec, typically the instructor. Once she is familiarized with the system, all she needs is a set of Java algorithms to generate a collection of animations with low effort. Effort required to construct each animation is limited to scheduling input/output data that adequately illustrate the recursive process and, optionally, to the graphical design of visualizations.

SRec allows the user interacting flexibly with animations. SRec allows tracing the execution of a recursive algorithm with the navigation power of a reversible debugger, but does not display some information relevant to debugging, e.g. the value of local variables. Also notice that the combined usage of step/jump actions allows making presentations where an algorithm is animated by only displaying relevant events [14].

Finally, SRec provides some auxiliary functions that facilitate its integration into a particular course [10]. It allows exporting animations in an external, standard format (GIF). In order to gain higher international diffusion, it supports multiple languages (currently, English and Spanish). A comprehensive web site is also available (http://vido.escet.urjc.es/Srec/index_en.html).

3.2 Use in Algorithm Courses

SRec is aimed at the comprehension and analysis levels of Bloom's taxonomy [1]. We illustrate here how SRec can assist in reaching these levels for several algorithm topics.

Firstly, the different customization options of SRec allow illustrating the behaviour of a recursive process with different levels of detail. Unfolding completely an activation tree displays its global behaviour. However, the *jump over* action, customized to not displaying underlying subtrees, allows highlighting the visible parts. In addition, an activation tree only displaying input values shows the recursive structure of its corresponding algorithm, whereas also displaying output values illustrates more accurately its behaviour.

SRec can also be used to illustrate the efficiency of recursive algorithms. Activation trees (with history) visualize the complete history of a recursive algorithm, so they are adequate to illustrate time efficiency. The execution stack visualizes the current state of computation, so it is adequate to illustrate space efficiency.

SRec facilitates understanding lineal recursive algorithms and therefore their conversion into iterative ones. Recursion removal is a key element of greedy algorithms, which are typically specified in a final recursive style and then converted into a loop.

SRec also is useful to analyze redundancy in multiple recursive functions (e.g. Fibonacci numbers). This analysis is necessary to obtain dynamic programming algorithms from recursive formulae. Alternatively, if a redundant algorithm is optimized by means of memoization, SRec allows checking that only calls invoked for the first time make further invocations.

Divide-and-conquer and backtracking algorithms may also be visualized with SRec. However, it comes to be less useful here since these algorithms produce shallow and very wide activation trees due to the visualization of data structures in each node. It is uncomfortable to read these displays, so alternative views or additional interaction techniques are required.

4. EXPERIENCE AND EVALUATION

SRec was used in Autumn 2007 in the course "Design and Analysis of Algorithms". The system was useful to the instructor during the classes to rapidly illustrate different issues of recursive algorithms. The instructor's impression was that the students' attitude was very positive. More objective results were obtained from a full usability evaluation described below.

4.1 Evaluation Procedure

The usability evaluation was made up of three steps [8]: a heuristic inspection (formative evaluation), an observational session and an empirical experiment session (summative evaluations). The details of the heuristic inspection are beyond the scope of the paper, so we describe here summative evaluations.

The observational session was performed in Spring 2007 in the optional course "Advanced Data Structures and Algorithms". Students were encouraged to participate, resulting in seven participants. We were able to observe this small number of students and to take notes of their activity. The empirical experiment session was performed in Autumn 2007 in the mandatory course "Design and Analysis of Algorithms". The session was a part of an optional assignment. A number of 28 students participated.

Each session was two hours long, structured in four tasks. Firstly, the instructor made a demonstration of SRec with an example. Then, the students were trained in using SRec with two additional examples. Finally, the students had to perform a given task. At the end, the students answered an opinion questionnaire about SRec, consisting in quizzes and open answers.

For the observational session, the students were asked to debug a version of the mergesort algorithm containing two bugs. As a result of this task, the students had to submit two deliverables: a document identifying the errors and explaining how they were fixed, and an animation of the debugged algorithm.

For the empirical experiment session, the students were asked to analyze a multiple recursion algorithm that solved a competition problem. They had to deliver a recursion tree and a dependency graph representatives of the algorithm. This task was a part of a larger assignment consisting in removing recursion redundancy.

4.2 Usability Results

The results of consulting the participants' opinion were very encouraging. All of the students of both sessions liked SRec. (Five students of the second evaluation did not answer this global question, but they answered the more specific questions.) All of the students in both sessions agreed in that SRec was easy to use. This opinion is reinforced by the complete lack of questions about the tool after the first task.

Most of the opinions about the quality of the main facilities of the tool were good or very good (47 out of 49 in the first session and 170 out of 224 in the second one). Students of the second session were more demanding because it was not isolated from the rest of the course but it was a part of an assignment. 51 grades were neutral, and only 3 features were marked with low quality. The highest scores were assigned to animation controls and activation trees. The lowest scores were assigned to secondary features of the system: icons of the user interface, customization facilities and interaction with panels.

Open answers and the observations of the instructors allowed identifying potential improvements and even new functions to enhance its usability. For instance, a new menu option allows re-processing source code after fixing bugs without the need to search and load the file from disk. Again, the students of the second evaluation made a higher number of suggestions.

4.3 Effectiveness Results

The first session showed the limitations of SRec. Six of the 7 students delivered their solution to the debugging task. Only 3 of them correctly identified, fixed and explained the bugs. All of the students declared that SRec assisted them in checking the correctness of their new algorithm, but the 3 successful students claimed that it did not assist them in debugging. In fact, we noticed during the observation that 2 of these students used BlueJ [9] to debug. We can explain this limitation: SRec was not design to debug, but to understand and analyze recursive algorithms. The debugging task was designed as an exercise to evaluate the usability of SRec, but instructors using the system must adhere to activities and assignments that fit its educational goals.

With respect to the second evaluation, all of the students delivered the visualizations requested in the fourth task. We marked right 26 out of 28 proposals. Of those students who did not pass the assignment, one delivered a recursion tree without redundancy and the other one mixed up the concepts. All of the students agreed with the high quality of SRec to analyze recursive algorithms. Only two students claimed that SRec did not assist them in analyzing the calls invoked in run time and only one claimed that SRec did not assist him in finding dependencies.

5. RELATED WORK

Programming environments support analysis of execution, but they hardly support specific analysis of recursion. Even visual programming environments give limited support to recursion [9]. We remark ETV [12], a tool that allows visualizing the execution of a C++ program from the trace generated in execution time.

Some systems were specifically designed to learn recursion. Most of them are based on the "copies model" of recursion, that is, they show a different copy of either code or data for each recursive call. We may cite *Recursion Animator* [15], SimRECUR [16], *Function Visualizer* [3] and EROSI [6]. Flopex 2 [4] is an Excel extension for visual programming. EROSI, *Recursion Animator* and SimRECUR have been successfully evaluated.

We have compared SRec with these systems according to different criteria. Table 1 shows a comparison of their generality and effort to construct animations, according to the description available in the literature. Several systems (including SRec) are automatic, being the most comfortable method to construct animations. SRec is the most flexible on selecting visualizable methods (it allows selecting any method). Finally, there are several systems (including SRec) that admit a wide range of types and input data.

Table 1. Comparison of generality and effort of several animation systems of recursion

System	Construction method	Selection of routine	Generality
EROSI	Predefined	Fixed menu	Given examples Bounded values
ETV	Automatic	Automatic (all unless otherwise stated)	Any types Any values
Flopex 2	Manual	Visual development	Any types Any values
Function Visualizer	Automatic	Automatic (first routine)	Any types Any values
Recursion Animator	Automatic	Automatic (first recurs. routine)	¿Types? ¿Values?
SimRECUR	Predefined	Fixed menu	Given examples ¿Values?
SRec	Automatic	Menu (one or several rout.)	Primitive types Any values

We also made a comparison with respect to their visualization and animation features. In summary, SimRECUR, ETV and SRec are the systems that offer more views and SRec has the most flexible set of animation controls.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have described SRec, an educational system to animate recursion. It is aimed at the comprehension and analysis levels of Bloom's taxonomy. SRec provides three coordinated views of recursion: traces, the execution stack and activation trees. It allows the user constructing and modifying animations without effort. The animation can be played flexibly, both forward and backwards. It also provides facilities to integrate animations into courses. We have described the educational features of the system and its use in algorithm courses. Finally, we performed a full usability evaluation and a comparison with other systems, obtaining very positive results in both cases.

We plan to work in three directions. Firstly, we are extending the educational facilities of SRec. Secondly, we want the system to exhibit more elaborate mechanisms to deal with large-scale recursive processes (e.g. overview+detail) [2]. Thirdly, SRec is the first step of a more ambitious project, aimed at animating algorithm design techniques. Thus, we are currently developing a system to animate divide-and-conquer algorithms.

7. ACKNOWLEDGMENTS

This work was supported by projects TIN2004-07568 of the MEC and S-0505/DPI/0235 of the CAM. We thank Carlos Lázaro his aid in the first session of SRec and Asunción Aldave for tabulating the results of the evaluations.

8. REFERENCES

[1] Bloom, B., Furst, E., Hill, W., and Krathwohl, D.R. 1956. Taxonomy of Educational Objectives: Handbook I, The Cognitive Domain. Addison-Wesley.

[2] Card, S.K., Mackinley, J.D., and Shneiderman, B. (eds.). 1999. Readings in Information Visualization. Morgan Kaufmann.

[3] Dershem, H.L., Parker, D.E., and Weinhold, R. 1999. A Java function visualizer. Journal of Computing in Small Colleges 15, 1 (1999), 220-230.

[4] Eskola, J., and Tarhio, J. On visualization of recursion with Excel. 2002. In Proc. Second Program Visualization Workshop. PVW'02. University of Aarhus, Denmark, 45-51.

[5] Fernández, L., Pérez-Carrasco, A., Velázquez-Iturbide, J.Á., and Urquiza-Fuentes, J. 2007. A framework for the automatic generation of algorithm animations based on design techniques. In LNCS 4753, 475-480.

[6] George, C.E. 2000. EROSI – Visualizing recursion and discovering new errors. In Proc. 31st SIGCSE Technical Symp. Computer Science Education. SIGCSE'00. 305-309.

[7] Haynes, S.M. 1995. Explaining recursion to the unsophisticated. SIGCSE Bulletin 27, 3 (Sept.), 3-6 and 14.

[8] Kerren, A., Ebert, A. and Meyer, J. (eds.). 2007. Human-Centered Visualization environments, Springer-Verlag.

[9] Kölling, M., Quig, B., Patterson, A., and Rosenberg, J. 2003. The BlueJ system and its pedagogy. Computer Science Education 13, 4, 249-268.

[10] Naps, T., Roessling, G., Cooper, S., Koldehofe, B., Leska, C. Dann, W., Korhonen, A., Malmi, L., Rantakokko, J., Ross, R.J., Anderson, J., Fleischer, R., Kuittinen, M., and McNally, M. 2003. Evaluating the educational impact of visualization. SIGCSE Bulletin 35, 4, 124-136.

[11] Stern, L., and Naish, L. 2002. Visual representations for recursive algorithms. In Proc. 33th SIGCSE Technical Symp. Computer Science Education. SIGCSE 2002. 196-200.

[12] Terada, M. 2005. ETV: A program trace player for students. In Proc. 10th Annual Conf. Innovation and Technology in Computer Science Education, ITiCSE 2005. 118-122.

[13] Velázquez-Iturbide, J.Á., Pareja-Flores, C., and Urquiza-Fuentes, J. 2008. An approach to effortless construction of program animations. Computers & Education 50, 1:179-192.

[14] Velázquez-Iturbide, J.Á., Redondo-Martín, D., Pareja-Flores, C., and Urquiza-Fuentes, 2008. J. An instructor's guide to design web-based algorithm animations. In Proc. ICWL'07.

[15] Wilcocks, D., and Sanders, I. 1994. Animating recursion as an aid to instruction. Computers & Education 23, 3, 221-226.

[16] Wu, C.-C., Lin, J. M.-C., and Hsu, I. Y-W. 1997. Closed laboratories using SimLIST and SimRECUR. Computers & Education 28, 1, 55-64.