PlanAni: A Program Animator Based on the Roles of Variables

Jorma Sajaniemi
Department of Computer Science
University of Joensuu

Marja Kuittinen
Department of Computer Science
University of Joensuu

Abstract

Computer programming is a difficult skill for many students and visualizations may be used to foster learning. This demo presents a program animation system, PlanAni, that is based on the concept of the roles of variables. Roles represent schematic uses of variables that occur in programs over and over again, and a set of nine roles covers practically all variables in novice-level programs.

CR Categories: K.3.2 [Computers and Education]: Computer and Information Science Education—Computer science education; D.m [Software]: Miscellaneous—Software psychology

Keywords: program animation, roles of variables

1 Introduction

Visualizations may be used to make both programming language constructs and program constructs more comprehensible [Hundhausen et al. 2002; Mulholland 1998]. Petre and Blackwell [1999] note that visualizations should not work in the programming language level because within-paradigm visualizations, i.e., those dealing with programming language constructs, are uninformative. Hence visualization of higher-level program constructs should be preferred to visualization of language-level constructs in teaching to program.

Sajaniemi [2002] has recently introduced the term role to describe the behavior of variables. For example, a generalization of the commonly known counter variables is the role stepper: a variable stepping through some systematic, predictable succession of values. Sajaniemi has found that only nine roles are needed to cover 99% of variables in novice-level programs.

Teaching elementary programming may be augmented by introducing roles to students. Traditionally, students have had to acquire this kind of higher-level knowledge from example programs and program fragments. Introducing the nine roles explicitly gives students a comprehensive set of concepts and a vocabulary that they can use in studying example programs and in authoring new programs.

In this demo we present a new program animation system, PlanAni, that uses variable roles as a basis for visualization.

PlanAni provides automatic program animation and it is intended to be used in teaching basic programming constructs to novices.

2 The PlanAni Program Animator

In PlanAni, each role has a visualization—role image—that is used for all variables of the role. Role images give clues on how the successive values of a variable relate to each other and to other variables. In addition to role images, PlanAni utilizes role information for role-based animation of operations, also. For example, Figure 1 gives visualizations for the two syntactically similar comparisons some_variable > 0. In case (a), the variable is a most-recent holder and the comparison just checks whether the value is in the allowed range. In the visualization, the set of possible values emerges, allowed values with a green background and disallowed values with red. The arrow that points to that part of the values where the current value of the variable lays, appears as green or red depending on the values it points to. The arrow flashes to indicate the result of the comparison.

In Figure 1(b) the variable is a stepper and, again, the allowed and disallowed values are colored. However, these values are now part of the variable visualization and no new values do appear. The values flash and the user can see the result by the color of the current value.

Just as comparisons are animated differently for different roles, the animation of assignment statements depends on roles. Figure 2 shows the animation of the assignment fib := fib+temp where fib is a gatherer. The old value of the variable moves to the lower left corner of the role image and becomes gray, the expression and a short arrow appear, the result of the expression appears inside the box, and finally the expression and the arrow disappear. Thus the animation stresses the typical behavior of gatherers: the new value is obtained as a combination of the old value and some other data.

Assignment to a stepper looks radically different: the numbers inside the role image scroll smoothly until the new value is between the middle footprints. Thus the animation stresses the fact...
that the new values of *steppers* are typically known as soon as the
succession of values starts.

Figure 3 is an actual screenshot of the PlanAni user interface
when the system is animating a simple program that checks whether
its input is a palindrome.

PlanAni is implemented using Tcl/Tk and it has been tested both
on Linux/Unix and Windows NT. The architecture consists of four
levels. The lowest level takes care of primitive graphics and animation, and implements the user interface. The next level knows
how to animate smallest actions that are meaningful to viewers of
the animation. This level is language independent in the sense that
it can be used to animate programs written in various languages,
e.g., Pascal, C, and Java. The next level takes as input a program to
be animated, annotated with the roles of variables and possible role
changes, and animates the program automatically. Finally, the uppermost level does not need role information because it finds roles automatically.

Currently, the two uppermost levels are not implemented. As a
consequence, animation commands must be authored by hand for
each program to be animated. Typically 5 animation lines are
required for each line in the animated program.

**Acknowledgments**

The authors would like to thank Mikko Korhonen and Ville Rapa
for programming PlanAni, and Anu Nevalainen and Aki Rissanen
for graphic design of the role images.

**References**

HUNDHAUSEN, C. D., DOUGLAS, S. A., AND STASKO, J. D.

MULHOLLAND, P. 1998. A principled approach to the evaluation
of SV: a case study in Prolog. In Software Visualization –
Programming as a Multimedia Experience, The MIT Press,

PETRE, M., AND BLACKWELL, A. F. 1999. Mental imagery in
program design and visual programming. International Journal
of Human-Computer Studies 51, 1–70.

in novice-level procedural programs. In Proceedings of IEEE
2002 Symposium on Human Centric Computing Languages and
Environments (HCC’02), IEEE Computer Society, 37–39.