An Empirical Analysis of Roles of Variables in Novice-Level Procedural Programs

Jorma Sajaniemi
University of Joensuu
Department of Computer Science
P.O.Box 111, FIN-80101 Joensuu, Finland
Jorma.Sajaniemi@Joensuu.Fi

Abstract

The use of all variables in 109 novice-level, but expert-written, procedural programs were analyzed in order to find a small but still comprehensive set of generic roles that describe the nature of the successive values a variable obtains. This paper gives the results of the analysis: a list of only nine roles that cover 99% of variables, frequencies of the roles, and discovered role changes.

1. Introduction

Programming involves the use of abstract concepts at various levels of abstraction. One such concept is the notion of variable plans which represent stereotypic uses of variables in programs. Ehrlich and Soloway [1] suggest that variable plans consist of such aspects as the variable’s role in the program, the manner the variable is initialized and updated, and a guard that may protect the variable against invalid updates. As examples of roles they give “counter variable”, “running total variable”, and “new value variable” (that holds the newest number given as input in a loop). Variable plans and roles are tacit knowledge that is not mentioned explicitly in teaching programming to novices. Rather, teachers tend to present programming language constructs and students have to acquire higher-level program constructs from example programs and program fragments.

In our study, we set forth to find a small but still comprehensive set of generic roles that would describe practically all variables in novice-level programming and could, thus, be used in teaching. We did this by looking at a large set of novice-level programs, and analyzed the use of each variable. We came out with a classification of nine roles that cover 99% of variables in the analyzed 109 programs.

2. Roles in novice-level programs

We started our analysis of the role concept with Ehrlich and Soloway’s [1] definition. They give three example roles that actually describe the goal that the variable has to fulfill. Rist [5] develops further this idea and defines a plan as a set of actions that achieve a goal, and a goal as a state to be achieved, e.g., to calculate a value or a series of values. Our idea of a role is in these terms a “dynamic goal” or an invariant that holds for a variable: the relation to other variables that the variable represents.

A more detailed list of roles is presented by Green and Cornah [3] in their proposal for a tool, Programmer’s Torch. Their roles are close to ours but include also other aspects, e.g., whether a variable is used to control the execution path. Our roles are based on the nature of the successive values a variable obtains, and we pay no attention to the way the values are further used.

To make a comprehensive list of roles in novice-level programs we analyzed all programs in three introductory Pascal programming textbooks[2, 4, 6]. The author of this paper first went through all the programs and created a classification of roles as new variables not fitting existing roles were encountered. After this phase, he wrote a short description for each role, and then another judge made an independent analysis of all the variables. The few cases of different classifications were discussed and the role descriptions were adjusted slightly but there were no problems in reaching mutual understanding.

In this analysis, arrays were treated as a single variable, and their role was usually determined by the roles of their elements, e.g., an array consisting of elements that all were most-recent holders was counted as a single most-recent holder. The only exception was the role organizer (to be described later) which describes the array as a whole.

Fields of record variables were treated as separate variables. Pointers were treated just like other variables; espe-
cally, the role of a pointer describes how the pointer behaves and not how the pointed variable behaves.

Dynamic variables and file variables were not included in the analysis. In procedures and functions, formal value parameters and local variables were treated as normal variables but formal variable parameters were unified with the actual parameter for the analysis.

The role descriptions are meant for human classifiers who are able to use their understanding of a program to capture the data flow through a variable and identify main phases of this flow. For example, a one-way flag may be reset to its initial value at the beginning of the main loop (and changed within a nested loop) yielding the one-way behavior within the main loop even though the flag goes both ways during the entire program execution. The identified roles are the following:

- **Constant**: a variable whose value does not change after initialization (e.g., an input value stored in a variable that is not changed later) possibly done in several alternative assignment statements (e.g., a variable that is set to true if the program is executed during a leap year, and false otherwise) and possibly corrected immediately after initialization (e.g., an input value that is replaced by zero if it is negative)

- **Stepper**: a variable going through a series of values not depending on values of other non-constant variables (e.g., a counter of input values, a variable that doubles its value every time it is updated, or a variable that alternates between two values) even though the selection of possibly alternative update assignments may depend on other variables (e.g., the search index in binary search)

- **Follower**: a variable going through a series of values depending on the values of a stepper or another follower but not on other non-constant variables (e.g., the “previous” pointer when going through a linked list, or the “low” index in a binary search)

- **Most-recent holder**: a variable holding the latest value encountered in going through a series of values (e.g., the latest input read, or a copy of an array element last referenced using a stepper) and possibly corrected immediately after obtaining a new value (e.g., to scale into internal data representation format)

- **Most-wanted holder**: a variable holding the best value encountered so far in going through a series of values with no restriction on how to measure the goodness of a value (e.g., largest input seen so far, or an index to the smallest array element processed so far)

- **Gatherer**: a variable accumulating the effect of individual values in going through a series of values (e.g., a running total, or the total number of cards in hand when the player may draw several cards at a time)

- **One-way flag**: a Boolean variable that can be effectively changed only once (e.g., a variable stating whether the end of input has been encountered) even though the new value may be re-assigned several times (e.g., a variable initialized to false and set to true each time an error occurs during a long succession of operations)

- **Temporary**: a variable holding the value of some other variable for a very short time only (e.g., in a swap operation)

- **Organizer**: an array which is only used for rearranging its elements after initialization (e.g., an array used for sorting input values)

- **Other**: all other variables

Table 1 gives the distribution of roles in the three books analyzed. The three most frequent roles cover 84.0% of all the roles, and only 1.1% of variables were classified as other with all of them occurring in the more advanced book.

Majority of steppers (77.8%) are always increased by 1 and reset to their initial values as appropriate. The next largest groups are steppers that are always decreased by 1 (5.1%), those that are always increased or decreased by 1 (7.6%), and pointers that go forward in a linked list (3.8%). The remaining 5.7% steppers have larger steps, but only 1.3% use multiplication or division to yield a non-constant step.

Most most-wanted holders (62.5%) find maximum of the value series they are watching. Other goodness criteria
used in the analyzed programs are minimum, and closeness to some special value.

Gatherers are most-often running totals (72.2 %) that sum up a series of values. The next largest group (16.7 %) use various combinations of addition and subtraction, e.g., calculating the sum of items when new items may be added and old items removed. Only 5.6 % combine addition and multiplication.

Variables classified as other contain arrays used to represent some data structure, e.g. stack, and two variables representing curious versions of some normal role not fitting to our definitions. As the number of other variables is very small and they represent more advanced concepts, we may deduce that the above set of roles is sufficient to characterize variables in procedural novice-level programming.

3. Role changes

Role changes are of two basic types: either the final value of the variable in the first role is used as the initial value for the next role (a proper change), or the variable is reinitialized with a totally new value at the beginning of the new role phase (a sporadic change). This distinction is not only technical but it is also meaningful for a human trying to comprehend a program.

In the analyzed programs there were 29 role changes, out of which only 5 were proper. The fact that proper role changes relate to only 0.9 % of variables implies that our set of roles covers typical novice-level program constructs well. Figure 1 shows role change types in the analyzed programs. The number of role changes on each arc is at most two except that there were 10 sporadic role changes from most-recent holder to stepper and 8 sporadic role changes the opposite way.

Two proper role change types, from most-recent holder to stepper or to gatherer, occur in cases where the initial value of some calculation (realized by the second role) is obtained as input and the input is guarded to avoid invalid values (realized by the most-recent holder role that allows the user to give new values until a valid one is obtained). In these cases there really are two phases in the lifetime of the variable: guarded input followed by the calculation. As the calculation can be of any type, it is natural to separate it from the input phase and, consequently, to have a role change.

The third proper change type is from gatherer to organizer. This cover cases where some day-based data is first gathered to an array to obtain totals, and the array is then sorted for output. Again, the array variable has two separate phases, both fitting nicely to our role list yielding more support to the adequacy of the classification.

4. Conclusions

We have analyzed novice-level procedural programs and found that the vast majority of variables can be characterized by nine different roles. The keys in keeping the role set comprehensive but small were to define the role concept to cover the sequence of successive values but to exclude the way variables are used, and to search for generic roles that represent fundamental relationships with other variables.

Acknowledgments

The author would like to thank Raija Sarajärvi for acting as the second judge in classifying variables.

References