UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series A

Roles of Variables From the Perspective of Computer Science Educators

Mordechai Ben-Ari and Jorma Sajaniemi

Report A-2003-6

ACM K.3.2, H.1.2
ISSN 0789-7316
ISBN 952-458-407-7

Roles of Variables From the Perspective of Computer Science Educators

Mordechai Ben-Ari and Jorma Sajaniemi

***Abstract*— Roles can be assigned to occurrences of variables in programs according to a small number of patterns of use that are both language- and algorithm-independent. Preliminary studies on explicitly teaching roles of variables to novice students have shown that roles are an excellent pedagogical tool for clarifying the structure and meaning of programs. This paper describes the results of an investigation designed to test the understandability and acceptability of the role concept and of the individual roles, as seen by computer science educators. The investigation consisted of a short tutorial on roles, a brief training session on assigning roles to variables, a test evaluating the subjects' ability to assign roles, and a set of open questions concerning their opinions of roles. The responses of 51 computer science educators were analyzed. Roles were identified by 85 % accuracy, and in typical uses of variables by 93 % accuracy. Subjects' comments on the role concept in general were mostly positive, and they believed that roles could contribute to understanding programs. The role set used in the investigation turned out to be suitable for describing variable usage in novice-level programs.**

I. INTRODUCTION

TEACHING programming involves more than just teaching the syntax and semantics of a programming language. The student must learn how the constructs of the language work together to implement the solution of a problem. This requires knowledge of algorithms, as well as knowledge of programming techniques and paradigms. The primary method of teaching programming is to present examples so that the students can generalize from the examples to general principles of problem solving, and it is often worthwhile to formalize this process of generalization, and to explicitly teach program design techniques.

The concept of *roles of variables* can be considered as another pedagogical technique within this tradition. In programming, variables are not used in an ad-hoc way; instead, there are a *few* patterns that can describe almost all the uses of variables. Variable roles are a concept that is different from the algorithmic patterns that are frequently used as pedagogical aids, because the concept of roles focuses on the data flow through single variables that is relatively independent of the algorithm, as well as independent of the programming language, though possibly not of the programming paradigm. For example, in the long list of patterns given in [1], we find Pattern D1 (The Counter Pattern Using a Loop) with the following structure (in C++):

```
while (cost != 0) {
    loopCount++;
    ...
}
```

and Pattern F1 (Performing an Action on Each Element of an Array) with the following totally different structure:

```
for (int index = 0;
         index < MAX_ELEMENTS;
         index++) { ... }
```

In terms of roles of variables, however, both `loopCount` and `index` take on a predictable sequence of values and are assigned the same role (*stepper*); it is easy to make this assignment just by examining the data flow. Since modern program design focuses on the importance of data and data flow, as opposed to control flow, we believe that classroom discussions based upon roles of variables can contribute to the ability of novice students to understand and write programs.

The concept of roles of variables is based upon earlier work on variable use. Ehrlich and Soloway [2] and Rist [3] were interested in the mental representations of variables, while Green and Cornah [4] wanted to help maintenance programmers by providing a tool that would explain the behavior of variables. Our approach to the role concept is to find a comprehensive, yet small, set of characterizations for variables primarily for use in teaching. In fact, some roles are typically taught in programming courses. For example, many textbooks introduce the role *counter* and its special case *loop counter*, and in a high-school textbook [5], two roles, called *counter* and *accumulator*, are explicitly taught and used in examples and exercises. Somewhat related to roles is the Hungarian notation [6], which is a convention to encode information into a variable name about its type, quantity, and quality. Several convention schemes exist and they are used in professional programming.

Roles of variables were identified by Sajaniemi [7] who analyzed programs in several introductory Pascal textbooks. He found nine roles that covered 99 % of variables in novice-level programs—a small enough set to be covered in programming teaching. Later, Kuittinen and Sajaniemi [8] conducted an experiment during an introductory programming course, which compared traditional teaching with teaching that used roles and role-based animation. The results of this experiment indicated that the introduction of roles improves program comprehension and program writing skills.

Mordechai Ben-Ari is with Department of Science Teaching of the Weizmann Institute of Science. Email: moti.ben-ari@weizmann.ac.il.
Jorma Sajaniemi is with Department of Computer Science of the University of Joensuu, Finland. Email: jorma.sajaniemi@joensuu.fi

TABLE I
DEFINITION OF THE ROLES IN THE INVESTIGATION

Fixed value	A variable which is initialized without any calculation and whose value does not change thereafter.
Stepper	A variable stepping through a succession of values that can be predicted as soon as the succession starts.
Most-recent holder	A variable holding the latest value encountered in going through a succession of values.
Most-wanted holder	A variable holding the "best" value encountered so far in going through a succession of values. There are no restrictions on how to measure the goodness of a value.
Gatherer	A variable accumulating the effect of individual values in going through a succession of values.
Transformation	A variable that always gets its new value from the same calculation from value(s) of other variable(s).
Follower	A variable that gets its values by following another variable.

II. PURPOSE OF STUDY

In this paper, we are interested in computer science (CS) educators' attitude to the role concept and individual roles: if CS educators do not find the role concept intuitive and easy to apply, it would be unrealistic to expect them to use roles in teaching. To reveal how CS educators react to the new concept, we conducted an investigation consisting of four phases: (a) a short tutorial on roles, (b) a training session on the assignment of roles to variables, (c) a test evaluating subjects' ability to assign roles, and (d) a set of open questions concerning subjects' opinions of roles. The investigation was designed to test the understandability and acceptability of the role concept and of the individual roles, as seen by CS educators. Furthermore, we wanted to see whether roles could be taught to CS educators using only a short tutorial.

III. THE ROLE CONCEPT

The role of a variable is defined according to the dynamic character of a variable embodied by the succession of values the variable obtains, and how the new values assigned to the variable relate to other variables. For example, in the role of a *stepper*, a variable is assigned a succession of values that is known in advance as soon as the succession starts. The role concept does *not* concern the way a variable is used in the program. A *stepper* is a *stepper* whether it is used to index elements in an array or simply to count the number of input values.

```
program doubles;
    var data, count, value: integer;
begin
    repeat
        write('Enter count: '); readln(data)
    until data > 0;
    count := data;
    while count > 0 do begin
        write('Enter value: '); readln(value);
        writeln('Two times ', value, ' is ', 2*value);
        count := count - 1
    end
end.
```

Fig. 1. Simple Pascal program

For example, consider the Pascal program in Figure 1, which contains three variables: `data`, `count`, and `value`. In the first loop, the user is requested to enter the number of values to be later processed in the second loop. The number is requested repeatedly until the user gives a positive value, and the variable `data` is used to store the last input read. The variable `value` is used similarly in the second loop: it stores the last input. There is no possibility for the programmer to guess what values the user will enter. Since these variables always hold the latest in a sequence of values, we will say that their role is that of *most-recent holders*. The variable `count`, however, behaves very differently. Unlike the other variables for which there is no known relation between the successive values, once `count` has been initialized, its future values will be known exactly: it will step downwards one by one until it reaches its limiting value of zero. The role of this variable is that of a *stepper*.

The definitions of the roles as used in this research is given in Table I. The *Roles of Variables Home Page* at http://cs.joensuu.fi/~saja/var_roles/ contains fuller explanations and examples.

The original definition given in [7] contained nine roles, as well as a role *others*, for a few cases that could not be fit into the other categories. The list of roles was obtained by analyzing all the programs in three introductory textbooks. Later, as a by-product of the present study, a new role (*transformation*) was identified and added to the role set.

The concept of roles of variables is concerned with the *deep structure* [9] of the program: Does the variable hold a predetermined sequence of values, for example, the values of the index of a for-loop? Or does it hold the best value encountered so far, for example, when searching for the largest value in an array? The *surface structure* of the program, primarily its syntactic structure, is much less relevant to the concept of roles. The name of the variable, the places where it occurs within an expression and the relation between the expression and the enclosing assignment and control statements are not important in assigning roles.

Even though roles have technical definitions, they are a cognitive concept. For example, consider a variable that takes on the values of the Fibonacci sequence by adding up pairs of previous values in the sequence. A mathematician can predict the sequence as clearly as a novice can predict the sequence of values of the index of a simple for-loop, so she may assign the role of *stepper*, because the values "can be predicted as soon as the succession starts." On the other hand, a novice who has never seen the Fibonacci sequence before may assign the role of *gatherer*, because the variable accumulates the previous values.

TABLE II
OCCURRENCES OF THE ROLES IN THE TRAINING PROGRAMS

Role	Program						Total
	1	2	3	4	5	6	
Fixed value	1	1		1			3
Stepper		2	1		1		4
Most-recent holder		1	2	1	1	1	6
Most-wanted holder				1	1	2	4
Gatherer			2				2
Transformation					1	2	3
Follower					2		2

TABLE III
OCCURRENCES OF THE ROLES IN THE ANALYSIS PHASE PROGRAMS

Role	Program						Total
	1	2	3	4	5	6	
Fixed value	1	2		1	1		5
Stepper	2	1		2	1		6
Most-recent holder			1			1	2
Most-wanted holder	1					2	3
Gatherer		1			1		2
Transformation		2	1				3
Follower					2		2
One-way flag				1			1

IV. METHODOLOGY

The investigation was conducted on the Internet. The research materials consisted of web pages divided into three phases. The *tutorial phase* introduced the concept of roles of variables, followed by a section for each role containing: (a) the definition of the role (as given in Table I), (b) a full sample program demonstrating the role, (c) additional examples of the use of the role, and (d) a list of additional properties that can assist in applying the role. The programs were written in Pascal, but the concept of roles of variables is not language-dependent (though it may be paradigm-dependent), so other imperative languages like C or Java could have been used. For brevity, full technical definitions of the roles were omitted; instead, the presentation was closer to the way roles could be introduced to novices learning to program.

For the purposes of this investigation, several simplifications were made in the overall concept, as well as in the details of the role definitions. Variables were limited to scalar and array types, though the roles can be applied to more complex types such as pointers and records, and parameters were not used in the example programs. All programs were also designed so that *role changes*, where the same variable takes on multiple roles in succession, did not occur. Three roles (*one-way flag*, *temporary* and *organizer*) accounting for only 5.2 % of all variables in the analysis in [7] were not included in the tutorial in order to simplify and shorten it; the new role *transformation* was added as noted above.

With these simplifications, the tutorial consisted of a single web page yielding 8 pages when printed, as the subjects were encouraged to do to improve readability.

Following the tutorial, subjects were presented with a *training phase*: this consisted of a sequence of six programs containing 24 variables taking on all of the roles described in the tutorial. The subjects were required to assign roles using radio buttons; to reduce the demands on short-term memory and to ensure accuracy in the use of the roles, each button label was linked to the corresponding role definition. After assigning roles to each variable in a program, the subject was given feedback on a separate page; his or her assignment of a role to each variable was compared with our canonical assignment. After studying the feedback, the subject could click on the link to the next program. Table II shows how many times each role occurred in the training programs.

The *analysis phase* was similar in format to the training phase; subjects were again presented with six programs containing 24 variables, but this time in a single web page. Upon assigning all of the roles, the results were sent by email to the authors. Subjects were also given the option to invent a new role by themselves or to indicate that they did not know which role to select, although they were encouraged not to do so. They also had the opportunity to append comments to their choices. If roles were not selected for all variables, a list of these variables was provided, and the subject could return to the original page to finish the task. Table III summarizes occurrences of roles in the analysis programs.

```
program saw;
const last = 7;
type  ArrayType = array [1..last] of integer;
var   value: ArrayType;   { Values to be checked          }
      i:      integer;    { Index of array                }
      up: boolean;        { Current direction is up?      }
      ok: boolean;        { Does saw property still hold? }
begin
      writeln('Enter ', last, ' values:');
      for i:=1 to last do read(value[i]);
      up := value[1] < value[2];
      ok := value[1] <> value[2];
      i := 2;
      while ok and (i < last) do begin
          ok := (up and (value[i] > value[i+1])) or
                (not up and (value[i] < value[i+1]));
          up := not up;
          i := i + 1
      end;
      write('Values '); if not ok then write('do not ');
      writeln('form a saw.')
end.
```

Fig. 2. Pascal program with controversial variables

While the roles in the training phase were straightforward, some of the variables in the analysis phase were "controversial," i.e., borderline cases. Such programs would not normally be shown to novices, but these variable usages were included in order to validate the definitions of the roles. One such program was an iterative program for constructing elements of the Fibonacci sequence discussed in Section III. Another (Figure 2) checks if a sequence of values forms a "saw," in which the direction of change of the values alternates. The controversial variables are `up` and `ok`. The former is ostensibly a stepper, because we can predict that its values alternate between `true` and `false`, though others roles are plausible since the initial value of the variable is computed from the values of other variables. The variable `ok` is a *one-way flag*—one of the three roles not included in the tutorial. A *one-way flag* is the role assigned to a variable that may change its value only once; this role is frequently used for a variable of type boolean used as a "flag." We wanted to see if the absence of this role would be missed by the subjects and what choices

TABLE IV
SUBJECTS' SELECTIONS FOR THE ROLES (PERCENT)

Role	n	Role selected									Total
		FIX	STP	MRH	MWH	GTH	TRN	FOL	OTH	DNK	
FIX	5	**91**		7		2					100
STP	6		**91**	2		1	4	2			100
MRH	2	7	1	**92**							100
MWH	3		1	1	**79**	3	3	10	1	3	100
GTH	2	1	1	10	1	**60**	26	1			100
TRN	3	9	1	7	3	1	**75**	4	1		100
FOL	2			2				**96**		2	100
OWF	1			8	10	10	61	6	**6**		100

they would make.

The first version of the material was pretested by using five CS educators as subjects. They used materials containing eight programs intended to form the analysis phase. The time needed to complete the task varied from 28 to 90 minutes with mode being 60 minutes. In order to create a task that the subjects would not find burdensome, the number of analysis phase programs was reduced to six so that the estimated time to complete the task would be less than one hour. We left out programs that posed no problems to the subjects, which resulted in a higher proportion of controversial variables in the remaining programs. Thus the final set has more occurrences of controversial roles than normally occur while teaching introductory programming. The results of the pretest also enabled us to clarify the explanations and examples. The final materials can be found at http://cs.joensuu.fi/~saja/role_survey/.

Subjects were recruited by publicizing the URL containing the research material among CS educators in the authors' countries, as well as on mailing lists belonging to the special interest groups in computer science education and psychology of programming. Subjects worked remotely on the web-based material at their own pace. They were instructed to start with the tutorial and training phase, and only then to perform the analysis phase. The initial page contained direct links to the various phases, so that there was no need to complete the whole task in a single session.

After assigning roles, subjects were asked for general comments on the role concept or specific roles. Moreover, they were asked to indicate their length of experience teaching introductory programming and/or advanced CS courses in high school and/or college or university. We asked for names and email addresses so that we could send them copies of the research results, though subjects were allowed to remain anonymous if they so wished. Subjects were also allowed to click on a "quit" link following the tutorial and the training phase; one subject used this form but did not leave any comments as to his or her reasons for quitting.

V. SAMPLE DEMOGRAPHICS

Fifty-three computer science educators volunteered to participate in the investigation. One subject selected a quit option while working on the material; another's result was discarded as apparently consisting of randomly selected answers. The results of the remaining 51 subjects were used in the analysis.

Subjects represented both high-school teachers ($n = 8$) and university or college teachers ($n = 38$). Some subjects had been worked at both levels ($n = 6$) while some did not report teaching at either level ($n = 11$). Figure 3 summarizes subjects' experience in teaching either introductory programming or advanced CS courses in either high-school or at higher level.

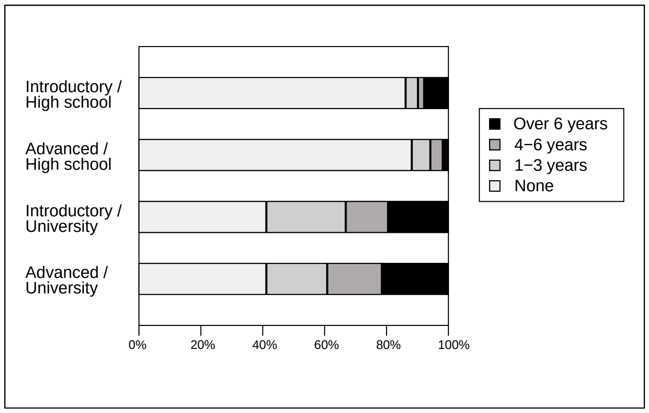

Fig. 3. Subjects' teaching background

VI. RESULTS

Table IV displays the selections made by the subjects for each role in the analysis phase. For each of the roles in the analysis programs, there is a row in the table labelled by that role, together with n, the number of variables having that role. The columns are labelled with the possible roles that could have been assigned. Therefore, an entry (r_1, r_2) in the table gives the percentage of the occurrences when role r_2 was assigned to a variable whose correct role was r_1. The diagonal, the percentage of correct assignments, is emphasized. For example, the first row describes subjects' selection for the five *fixed values* in the analysis programs. In 91 % of cases, subjects' selections were correct, while in 7 % of cases *most-recent holder* and in 2 % of cases *gatherer* was selected. OTH means that a subject thought the variable to have some *other* role not listed in the tutorial, while DNK means that a subject *did not know* which role should be used. OWF is the role *one-way flag*, discussed below.

Most roles are identified by at least 90% accuracy. The low identification of *most-wanted holders* and *gatherers* can be explained by controversial variables. In non-controversial cases *most-wanted holders* were identified correctly in 91% of the cases, and *gatherers* in 94% of the cases. Only 75% recognized *transformations*, though even here, in a simple case the role was recognized with 90% accuracy.

Figure 4 contains a frequency graph depicting the distribution of subjects' error scores including the four controversial variables. The assignments to the occurrence of the *one-way flag* were not used in computing these error rates, because the role had not been presented to the subjects in the tutorial. There were an average of 3.4 errors in the classification of the remaining 23 variables (i.e., average accuracy of 85 %), and an average of 1.3 errors in the classification of the 19 non-controversial ones (i.e., average accuracy of 93 %),.

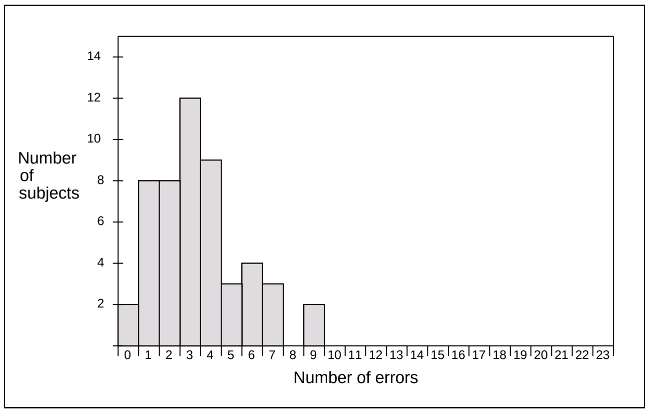

Fig. 4. Frequencies of error scores

The mean error score for subjects having more teaching experience in high-school than colleges or universities was 4.67 while it was 3.18 for subjects having more experience in colleges or universities (two-tailed t test, $t = 1.571, df = 38, p = .0175$). Figure 5 shows teaching experience in colleges or universities for the high performers (at most 1 error, $n = 10$) and low performers (at least 5 errors, $n = 12$). High perform-

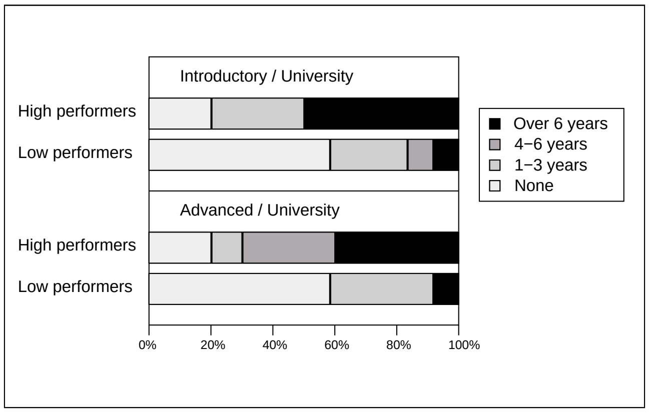

Fig. 5. High and low performers' university teaching background

ers are more experienced both in introductory programming courses (two-tailed t test, $t = 2.310, df, p = .0317$) and in advanced CS courses ($t = 2.944, df, p = .0080$). There is a similar trend with regard to high-school teaching experience, but this is not statistically significant.

We also looked at the distribution of error selections for high and low performers. To obtain roughly the same amount of errors for both groups, high performers were now defined to be subjects with at most 3 errors (making a total of 60 errors) and low performers were subjects having at least 6 errors (with a total of 63 errors). We counted the number of erroneously selected roles for each variable; for all variables (except the *one-way flag*) low performers selected a wider variety of roles. The difference is statistically significant (paired t test, $t = 3.943, df = 22, p = .0007$). For the *one-way flag* the difference is opposite: high performers made a wider variety of selections (5 different roles) than low performers (3 different roles).

Results of the open questions will be discussed in Section VIII.

VII. Error analysis

We begin with an analysis of the errors, distinguishing between the high and low performers. The errors are analyzed in terms of deep vs. surface structure, as well as of typical vs. atypical use of the variables.

A. *Surface vs. deep structure*

Recall from Section I that roles of variables are used to describe the deep, rather than the surface, structure of the program. When analyzing errors this distinction is important: for example, an error that confuses two roles with different deep structures but similar surface structures reveals that the subject has a weak understanding of the distinction between two roles. This difference between surface and deep structures was explicitly mentioned by one of the subjects who was analyzing the role of the variable `second` that holds the second largest value seen so far in a search for two largest values:

> At the code [i.e., surface] level `second` seems to have two roles "follower," and "most-wanted"), but semantically [i.e., at the deep level] it has just one: "most-wanted." It is the best choice for the criterion "to be the 2nd largest."

B. *Atypical use of variables*

Errors in the assignment of roles were frequently caused by *atypical* use of the variables. For example, *steppers* are typically used for sequences whose values increase or decrease monotonically, often in an arithmetic progression. This typical use is so dominant that it may be hard to recognize that other successions, such as the alternating sequence of values $\{$`true`, `false`, `true`, `false`, $\ldots\}$ assigned to the variable `up`, are just as predictable and therefore this variable should also be assigned the role *stepper*. Many subjects erred, because although the surface structure is typical (`up:=not up` is syntactically similar to the typical `i:=i+1`), the deep structure with its nonmonotonic sequence is not. Being misled by an atypical use of a variable does not indicate that the role is counterintuitive; instead, it is a pedagogical challenge to learn to gloss over surface structure and analyze deep structure. The controversial variables presented in the investigation involved uses with both atypical surface and deep structures. The *most-wanted holder* `second` has an atypical surface structure, while the *stepper* `up` has an atypical deep structure.

C. Errors made by high performers on non-controversial variables

Errors made by high performers on non-controversial variables are potential indicators that the entire concept is not viable. There were only two error types of this kind.

Some subjects had difficulty in assigning roles to variables that are arrays. Forewarned on this issue by the pretest, we emphasized this point in the tutorial:

> [N]ormally all elements of an array have the same role, e.g., an array of *fixed values* or an array of *most-wanted holders*; in this case the role of the array is that of its elements, i.e., the array as a whole is considered to be a *fixed value* or *most-wanted holder*, respectively.

Nevertheless, several subjects assigned *most-recent holder* to an array of *fixed values*, because, considered as a sequence of individual variables, each component of the array holds the latest value read from the input. The error was less frequent on the subsequent occurrence of an array in a program, so we believe that with more experience, subjects would cease to make the error; furthermore, exercises in assigning roles to array variables should also reduce the frequency of this error.

The second error concerned the role of *transformation* which identifies cases where a variable has no independent existence, but merely serves to contain a value obtained by computation, for example, a unit conversion from a number to a percentage, or from degrees to radians, or a split of a number into its quotient and remainder upon division by another number. The difference between a *transformation* and the role of the original variable, or between a *transformation* and some other role having the same surface structure, was not always apparent to the subjects. For example, the variable `factor` is a transformation computed as `percent/100`, but thereafter not modified. Clearly subjects were justified in assigning the role *fixed-value* to that variable.

D. Errors made by high performers on controversial variables

There were also two types of errors made by high performers when assigning roles to controversial variables.

The first type appeared in three cases when an atypical surface structure misled some subjects to select the role whose typical surface structure matched the variable in question. For example, if the variable was assigned different values in the two alternatives of an if-statement, some subjects did not recognize that the variable was still assigned a value only once and thus should be assigned the role *fixed-value*. In another case, `fibN`—the “current” number in the Fibonacci sequence—was assigned *most-recent holder* which corresponds to its surface structure rather than *gatherer* which corresponds to its deep structure.

The second error type (one case) was caused when an atypical deep structure triggered a large variety in the roles suggested by the high performers. In this case (the variable `up` discussed above), the subjects searched for roles with a more appropriate deep structure. Since the roles are designed to characterize distinct deep structures, there were no others that were appropriate for this case resulting in a variety of answers.

E. Errors made by low performers

In general, low performers made the same types of errors for both controversial and non-controversial variables as did high performers and made them more often. Other errors that they made can be explained by a tendency to make decisions based on surface structure only. The role *most-wanted holder* was assigned even when no possible “measure of the goodness of the value” existed; *gatherer* was assigned when no “accumulation” was being carried out; *follower* was assigned when it did not contain values of the variable being followed, etc.

An interesting exception to the similarity of errors is the case of the controversial variable with atypical deep structure (the variable `up`): high performers spread their errors evenly among many roles, while low performers concentrated their errors on a single role *transformation*, the one with closest surface structure. The same behavior was seen in the case of the *one-way flag*, which was the role that was missing from the tutorial. High performers assigned a larger variety of roles than did the low performers. When confronted with a variable where none of the roles seems to appropriate, low performers tend to look at the surface structures. In this case, the structure of the expression matched a few roles only, so the low performers made their selection among these. On the other hand, high performers look at the deep structures, and, as discussed above, found the other roles equally inappropriate.

F. Other errors

The roles *most-recent holder* and *transformation* were most often assigned in case of doubt, presumably, because their definitions are the least specific. Any variable holds the most recent value of some calculation, even though *most-recent holder* is reserved for “raw” data such as input values. Similarly, many variables are the result of computation from other variables, even though *transformation* is intended to be used in specific cases like scaling values.

Some subjects detected spurious role changes. Since the web form was designed with radio buttons so that each variable could only be assigned a single role, these difficulties were indicated in the comments. Two types of reasons can be recognized for these errors:

- Role definitions are not meant to be applied too literally. For example, some subjects assigned a role to an atypical initialization of a variable, followed by a role change for the use of the variable.
- A variable may be used after it has ceased to change, so some subjects believed this to be a role change to a *fixed value*, in particular, if the variable was used within a loop.

VIII. Discussion

We will now discuss the findings and subjects’ answers to open questions as regards to the purpose of this study: Do CS educators find the role concept intuitive? Are the individual roles understandable and easy to recognize? Can roles be taught to CS educator using only a short tutorial.

A. Learnability and acceptability

We did not ask the subjects to report the amount of time spent on the task, but based on the pretest, we believe that this should be less than one hour. The fact that subjects agreed with our assignment of the roles after such a short introduction is encouraging; in more than 90% of the cases for non-controversial variables for every role except *transformation*. The results make it clear that increased teaching experience improves performance, and they indicate that experts have little problem with the role concept, supporting the assumption that roles represent tacit expert knowledge. Roles are not hard for non-experienced teachers either, since for the 19 non-controversial variables, 86% of the subjects made at most two errors.

Many subjects stated in their comments that they had had problems in remembering the definitions of the roles or that the definitions were ambiguous, but the same subjects scored between zero to two errors on non-controversial variables, indicating that they understood, perhaps subconsciously, the deep structure of variables represented by the roles. Subjects' comments on the role concept in general were mostly positive, and they believed that roles could contribute to understanding programs. Some subjects, even among the high performers, were skeptical about the importance of roles in teaching. One subject was uncertain about the interaction of roles with data flow and invariants, while another raised the possibility that they may interfere with learning other programming paradigms.

B. Suitability of the role set

In general, the role set used in the investigation seems to be suitable for its purpose. The naturalness of the roles is further supported by the decrease in the number of errors with increasing experience. Even when unsure, subjects' assignments of roles was quite good.

The *one-way flag* role that was not included in the tutorial garnered the largest number of alternative suggestions for other roles. This provides evidence that the *one-way flag* is a distinct role, not subsumed or similar to the others. New roles suggested by the subjects, including *checker*, *guardian*, *state* and *latch*, are consistent with our definition of the role. Furthermore, the dearth of suggestions for new roles for the other variables supports our claim that the role set is sufficient for the analysis of variables in novice-level programs.

The only role that caused frequent confusion was *transformation*. As discussed above, it is intended to identify cases where a variable has no independent existence, but merely serves to contain a value obtained by computation. In a sense, this role "usurps" the role or roles assigned to the variables from which the transformation is computed. In the case of `factor:=percent/100`, the *transformation* role usurps the *fixed-value* role that would normally be assigned to `factor`. The definition of this role has to be clarified.

Variables become controversial if either the surface structure or the deep structure is atypical. The ability to recognize and go beyond an atypical surface structure is gained by increased expertise and developing this ability is the task of the teacher. Variability in surface structures is so large that it makes no sense to add or modify roles to take surface structure into account. On the other hand, atypical deep structure is a sign that a new role might be needed. This claim is justified by the similarity in the subjects' approach to the atypical deep structure and to the missing role. Since we want to keep the number of roles small so that they can be used in introductory teaching, we prefer that new roles that rarely occur should be embedded within the existing roles.

C. Tutorial and training materials

The tutorial and training materials were deliberating kept short to encourage compliance by our subjects. Obviously, the number of examples should be much greater in order to explore the ramifications of the definitions of the roles in a wider selection of programs. This will not be a problem in an educational setting, where the roles can be introduced gradually during an introductory course and reinforced in all the examples and exercises. On the other hand, subjects made errors in many cases explicitly covered in the tutorial. Thus, we should not expect that improving the tutorial would automatically yield a better performance. It is therefore important that the subjects find the roles easily understandable and natural.

Subjects made only a few comments on the tutorial and training programs, mostly concerned with details of the material, or with issues like role changes that were deliberately left out. There were, however, some suggestions for the inclusion of role interactions and conflicts, and additional guidance for role usage to construct clear programs.

The error analysis suggested some improvements for the tutorial. First, the tutorial should stress that roles concern the deep structure of variables, even though roles can often be identified from typical surface structures (for example, the index of a for-loop is almost certainly a *stepper*). Second, the tutorial has to give criteria for distinguishing *transformations* and *gatherers* from other "computationless" roles. Third, role changes—that were skipped in the investigation—must be introduced in a way clearly allowing "extra" steps in the initialization of variables. Finally, the application of roles to structured data types must be covered thoroughly.

IX. Conclusion

The concept of roles of variables can be used as a pedagogical technique to teach how the constructs of a programming language work together to implement the solution of a problem. Preliminary results of using roles in teaching elementary programming indicate that the introduction of roles improves program comprehension and program writing skills. In this paper, we were interested to find out how computer science educators react to this new concept and to the individual roles. The outcome of the investigation is encouraging, because CS educators accepted the concept of roles as intuitive and find it easy to assign roles consistently.

Even in those cases where assignment is controversial, the debate itself can be an excellent pedagogical tool for clarifying the structure of programs in introductory courses. It is important to emphasize that we do not regard roles as an end in themselves and we do not think that students should be graded on their ability to assign roles. Roles of variables are design rules and pedagogical aids intended to help novices over the hurdle of learning programming.

Roles of variables can also change the way that program visualization and animation are carried out [10]. Traditional systems such as Jeliot [11] provide visualizations that operate on the programming language level; therefore, the representation and animation of variables is uniform reflecting the surface structure of the program, not its deep structure. Role-specific representation of variables and role-specific animation for operations should result in visualizations on a higher level that will be more informative to students.

The error analysis revealed the importance of making the distinction between surface and deep structures of the roles. Typical uses of the roles were recognized easily, but the error behavior was different depending on the level of the atypical features. In particular, the effect of atypical deep structures was similar to the effect of the missing role. This suggests a method for extending the concept of roles to other programming paradigms, for example, to object-oriented programming: analysis of atypical deep structures of variables in real programs would indicate the need for additional or modified roles. Since atypical surface structures were found to be problematic for low performers, learning materials must explicitly address this issue.

Future research on roles will include: (a) cognitive studies to investigate if roles are truly part of the knowledge structure of experts, though even if the answer is negative, it would not rule out their pedagogical use; (b) further empirical research in classrooms in order to determine if roles are pedagogically useful; (c) further development and evaluation of program animation for visualization of roles; (d) extension of the role set to cover other programming paradigms and other expertise levels.

Acknowledgments

We would like to thank all those who volunteered to participate in the study, and especially Noa Ragonis for her extensive comments on the first version of the tutorial and on a draft of this paper.

References

[1] C. Sollohub, "C++ in Hypertext," 2001, http://cs.nmhu.edu/personal/curtis/cs1htmlfiles/Cs1text.htm.

[2] K. Ehrlich and E. Soloway, "An empirical investigation of the tacit plan knowledge in programming," in *Human Factors in Computer Systems*, J. C. Thomas and M. L. Schneider, Eds. Norwood, NJ: Ablex Publishing Co, 1984, pp. 113–133.

[3] R. S. Rist, "Knowledge creation and retrieval in program design: A comparison of novice and intermediate student programmers," *Human-Computer Interaction*, vol. 6, pp. 1–46, 1991.

[4] T. R. G. Green and A. J. Cornah, "The programmer's torch," in *Human-Computer Interaction—INTERACT'84*. IFIP, Elsevier Science Publishers (North-Holland), 1985, pp. 397–402.

[5] D. Ginat, *Foundations of Computer Science 1*. Rehovot, Israel: Weizmann Institute of Science, 1999, (in Hebrew).

[6] C. Simonyi, "Hungarian notation," 1999, http://msdn.microsoft.com/library/en-us/dnvsgen/html/hunganotat.asp.

[7] J. Sajaniemi, "An empirical analysis of roles of variables in novice-level procedural programs," in *Proceedings of IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02)*. IEEE Computer Society, 2002, pp. 37–39.

[8] M. Kuittinen and J. Sajaniemi, "First results of an experiment on using roles of variables in teaching," in *EASE and PPIG 2003, Papers from the Joint Conference at Keele University*, 2003, pp. 347–357.

[9] F. Détienne, *Software Design – Cognitive Aspects*. London: Springer Verlag, 2002.

[10] J. Sajaniemi, "Visualizing roles of variables to novice programmers," in *Proceedings of the Fourteenth Annual Workshop of the Psychology of Programming Interest Group (PPIG 2002)*, J. Kuljis, L. Baldwin, and R. Scoble, Eds., 2002, pp. 111–127.

[11] M. Ben-Ari, N. Myller, E. Sutinen, and J. Tarhio, "Perspectives on program animation with Jeliot," in *Software Visualization: International Seminar*, ser. Lecture Notes in Computer Science 2269, Dagstuhl Castle, Germany, 2002, pp. 31–45.