A Web-Based Service for the Automatic Detection of Roles of Variables

Petri Gerdt  
Department of Computer Science  
University of Joensuu  
Joensuu, Finland  
pgerdt@cs.joensuu.fi

Jorma Sajaniemi  
Department of Computer Science  
University of Joensuu  
Joensuu, Finland  
saja@cs.joensuu.fi

ABSTRACT

Learning elementary programming can be enhanced by introducing the notion of variable roles to students. This paper presents a web-based automatic role detection service that can be utilized in teaching when consolidating role knowledge. The service is based on data flow analysis techniques and uses machine learning to create flow characteristics for roles. The current version recognizes roles with 93% accuracy which is comparable to that of human role assigners.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education—computer science education; D.m [Software]: Miscellaneous; F.m [Theory of Computation]: Miscellaneous

General Terms

Human factors

Keywords

Roles of variables, automatic detection of roles, role knowledge consolidation

1. INTRODUCTION

Roles of variables have been found to enhance learning elementary programming [3, 13]. Roles increase students’ programming knowledge and the possession of role-related variable schemas makes programming fluent [3]. Familiarity with roles also changes students’ mental representations of programs so that they become similar to that of good program comprehenders [13]. Thus roles have been found to give novices better programming skills and better program comprehension capabilities.

Roles are a part of expert programmers’ knowledge [14] and they give a classification of various behavior patterns for variables. For example, a variable that steps through a predefined series of values, say 2, 4, 6, . . . has the role of a stepper—a generalization of the notion of a counter. Roles apply to variables, attributes and parameters in object-oriented, imperative and functional programming and a set of eleven roles covers practically all variables in elementary courses (i.e., CS1/2) [11]. Roles that were used in the study reported in this paper are fixed value, stepper, most-recent holder, most-wanted holder, gatherer, follower, transformation, one-way flag, and temporary. For a more comprehensive treatment of the role concept, see the Roles of Variables Home Page at http://www.cs.joensuu.fi/~saja/var-roles.

Roles are cognitive concepts and cannot be given exact, formal definitions. In some cases, people may even see the behavior of a certain variable differently and hence disagree of its role. When the assignment of a role is controversial, the debate itself can, however, be an excellent pedagogical tool for clarifying the structure of programs in introductory courses. 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. Instead, roles are design rules and pedagogical aids intended to help novices over the hurdle of learning programming.

When roles are used in learning, improvements in programming skills are achieved only if role knowledge is consolidated so that its application becomes autonomous. For example, the use of a role-based program animator elaborated students’ role knowledge so that they could apply roles in programming fluently; benefits to students that used a traditional debugger for program animation were much smaller [3]. Another way to elaborate role knowledge is to write roles in comments attached to variable declarations. As a result, students learn to think of variables in their own programs as having some special behavior that enables the variable to fulfill its task in the program. In program comprehension, students may be asked to assign roles as the first step in understanding the behavior of the whole program. Thus role knowledge consolidation may serve the introduction of program writing and program comprehension strategies.

When assigning roles, students may at first be unsure about the differences between roles, so assistance in role detection can be needed. In program comprehension, automatic assignment of roles can speed up the comprehension process and make the associated comprehension strategy—find first the roles and then their interaction—more visible. For such purposes we have implemented an automatic role detection system and its web-based user interface, the Role
Advisor, which is the topic of this paper. In addition to assisting in role detection, the Role Advisor can be used to demonstrate the general applicability of roles: given any program—even one that has not been designed with roles in mind—it will reveal the roles and thus point out that roles describe inherent properties of all variable use.

A somewhat similar system is being developed by Bishop and Johnson [2]. Their system is intended to be used for checking students’ role assignments in simple Java programs and it is implemented as a BlueJ plugin. Even though the basic role detection techniques—notably the use of data flow analyses—are common with our system, their system differs from ours in several respects. First, their system is used within the BlueJ programming environment whereas our system is a web-based service. Second, because the BlueJ plugin is intended for checking students’ role assignments, the system gets as input both a program and the student’s suggestions for roles. The system figures out whether each of the student’s assignments is reasonable, and if not, provides an explanation of why the student’s assignment is inappropriate. Our system takes no student suggestions but determines roles based on the program only. Third, their system uses fixed role definitions whereas our system recognizes the cognitive nature of roles by using machine learning techniques in acquiring role definitions. Our system will adapt itself into new definitions of roles if role assignments in learning materials are changed.

The rest of the paper is organized as follows. Section 2 describes the Role Advisor from a users’ point of view and Section 3 explains the automatic role detection mechanism. Section 4 presents an evaluation of the detection mechanism and, finally, Section 5 contains the conclusions.

2. THE ROLE ADVISOR

The user interface of the Role Advisor consists of two web pages. On the first page (Figure 1) the user may enter the program to be analyzed and use radio buttons to ask for the roles of all variables in the program or of a single variable only. There are also links to instructions, example programs that can be used for testing the interface, list of roles with informal definitions and examples, etc. Thus the first page provides links to basic informations about roles in addition to the input fields.

The second page (Figure 2) presents the results of the automatic role analysis. For each variable, the Role Advisor lists its name and role. The role name is a link to the corresponding role definition in the role list that opens into another window. The name of the variable can also be clicked. A click on the variable highlights all occurrences of the variable in the program text that is also visible on the page. Thus a user may see all the statements affecting the behavior of the variable as well as for what purposes the variable is used in the program. In Figure 2, the variable previous is highlighted this way.

The text on the second page stresses that different people may assign different roles to the same variable so that students would not be oppressed if the Role Advisor’s suggestions do not correspond to their interpretations. Students are encouraged to discuss differences in role assignments with their teachers—a source of meaningful discussions in learning sessions.

When teaching program comprehension strategies, students can be instructed to start the comprehension task by asking the Role Advisor for the roles of all variables. Then, students can study the program code in the user interface, and reveal the meaning of individual variables by the highlighting technique. Letovsky [8] has found that questions concerning the meaning of variables (so called “what” questions) are asked as soon as the unfamiliar object is encountered and that such questions require a rapid answer in order to make conjectures about the goal (“why”) of an action possible. The Role Advisor provides rapid answers to such typical questions in program comprehension. With a suitable set of comprehension tasks, students may be introduced not only to comprehension strategies but also to typical solutions for frequently occurring programming problems.

In programming tasks, students can use the Role Advisor to analyze roles in their own programs. Any deviations from
can be run in two different modes. In the
program [9]. The RAE is a machine learning system that
can be used to make estimations of the run-time behavior of
analysis techniques—a collection of compile-time techniques that
the user interface script is fairly trivial.

This section concentrates on the RAE, as the implementation of
is implemented as a PHP script which gives the program
to be analyzed to the RAE. The user interface script then
displays the results provided by the RAE. The rest of this
section concentrates on the RAE, as the implementation of
the user interface script is fairly trivial.

In order to detect roles, the RAE uses static program anal-
ysis techniques—a collection of compile-time techniques that
can be used to make estimations of the run-time behavior of
a program [9]. The RAE is a machine learning system that
can be run in two different modes. In the learning mode, it
is provided with role-annotated programs, which it uses to
learn what kind of run-time behavior of a variable accounts
for each role. In the matching mode, it assigns roles to the
variables of a program that has no role annotations.

Figure 3 displays the steps (represented by gray rectan-
gles) of the process in both modes. An arrow indicates the
flow of information from one step to the next one; a dashed
arrow denotes information flow that is present in the learn-
ing mode only. The first four steps are identical in both
learning and matching; only the final step is different be-
tween the two modes.

The first step in the detection process is the scanning and
parsings of the program, which results in an abstract syntax
tree (AST). An AST is a tree representation of a program;

3. THE ROLE ADVISOR ENGINE

The implementation of the Role Advisor consists of two
main parts: (1) the user interface and (2) the Role Advisor
Engine (RAE), which detects the roles. The user interface
is implemented as a PHP script which gives the program
to be analyzed to the RAE. The user interface script then
displays the results provided by the RAE. The rest of this
section concentrates on the RAE, as the implementation of
the user interface script is fairly trivial.

In order to detect roles, the RAE uses static program anal-
ysis techniques—a collection of compile-time techniques that
can be used to make estimations of the run-time behavior of
a program [9]. The RAE is a machine learning system that
can be run in two different modes. In the learning mode, it
is provided with role-annotated programs, which it uses to
learn what kind of run-time behavior of a variable accounts
for each role. In the matching mode, it assigns roles to the
variables of a program that has no role annotations.

Figure 3 displays the steps (represented by gray rectan-
gles) of the process in both modes. An arrow indicates the
flow of information from one step to the next one; a dashed
arrow denotes information flow that is present in the learn-
ing mode only. The first four steps are identical in both
learning and matching; only the final step is different be-
tween the two modes.

The first step in the detection process is the scanning and
parsings of the program, which results in an abstract syntax
tree (AST). An AST is a tree representation of a program;

Each variable goes through a sequence of values during its
lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
ing its lifetime. This sequence can be described by flow charac-
teristics (FCs), which characterize the properties of the sequence
of values or the data flow that the variable encounters dur-
in the matching mode, the fifth step (“5b: Matching”)...
starts with the creation of role profiles. A role profile is a generalization of all instances of role–FC descriptions of a certain role in the role–FC database. The frequency of an FC in a role profile is the percentage of role–FC instances that include the FC of all instances of that certain role. Thus, each role profile has a frequency value for each FC.

The FCs are then divided into three categories by their frequencies. The categories are required FCs, tolerated FCs, and non-tolerated FCs. The division requires two parameters, the upper boundary and the lower boundary. The upper boundary indicates the minimum frequency that is classified as a required FC. Similarly, the lower boundary indicates the maximum frequency that is classified as a non-tolerated FC. An FC, whose frequency is between the lower and the upper threshold, is classified as a tolerated FC. The boundary parameters depend on the role–FC database, and they are automatically adjusted so that each role profile is unique, i.e., each role is mapped to a unique set of FCs. If the programs that are used in the learning mode are very coherent in respect to variable behavior then the upper boundary is close to 100% and the lower boundary is close to 0%.

Finally, the role of a variable is determined by comparing its variable profile to each of the role profiles and calculating distances between the variable profile and the role profiles. The distance is calculated simply by adding one for each required FC that is not present in the variable profile and one for each non-tolerated FC that is present in the variable profile. The presence or absence of tolerated FCs does not affect the distance score. The role profile that has the least distance is the matching profile, i.e., the suggested role of the variable. It is possible that multiple role profiles have the same distance scores for some variable profile; in that case the RAE suggests multiple roles for the variable.

The RAE is implemented in Tcl. It is built with the help of Yeti [10], a Yacc-like compiler-compiler [1].

4. VALIDATION

To assess the quality of the automatic role detection process, role assignments suggested by the RAE were compared with human role assigners. The material for the validation consisted of example programs in three Pascal textbooks: two elementary programming books intended for novices [4, 12] and one dealing with more advanced programming [5]. The variables in the example programs were assigned roles by researchers.

The performance of the RAE was examined with single cross-validation, where a single data set is split into two parts: the first part is used to formulate a model and the second part to evaluate the model [7]. In our case the data set consisted of all of the example programs mentioned above, and the model is the role–FC database together with the algorithms used for automatic role detection. The first part of the data set, which was used to construct the model is called learning material, and the second part that was used for validating the model is called matching material.

The learning material was created by selecting every other example program from each of the textbooks; the remaining programs made the matching material. This selection method ensures that both the learning material and the matching material are approximately of the same complexity. The programs that were used for learning were annotated with role information; the matching material did not include any role information. The role–FC database was built by running the RAE in the learning mode with the learning material as input.

There were originally 110 programs, but 23 programs that dealt with pointers or contained procedures were omitted, as the current RAE version does not implement pointer analysis and restricts itself into intraprocedural analysis. The current version does not handle arrays or role changes either, so 55 arrays were omitted and 10 variables with role changes were replaced by several variables—one for each role. As a result the learning material consisted of 43 programs with 175 variables and the matching material consisted of 44 programs with 130 variables.

Table 2 gives for each role the number of variables in the learning material (nlearn) and in the matching material (nmatch). The last column gives the percentage of correct role suggestions, i.e., the proportion of variables in the

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional definition (COD)</td>
<td>The variable has a definition that is done in a branch of a conditional structure.</td>
</tr>
<tr>
<td>Loop assignment (LOA)</td>
<td>The variable has an assignment statement that is done within a loop structure.</td>
</tr>
<tr>
<td>Conditional expression participati-</td>
<td>The variable participates in a conditional expression that may affect the value sequence of it.</td>
</tr>
<tr>
<td>(CEP)</td>
<td>Loop expression participation (LEP) The variable participates in a loop expression that affects the value sequence of the variable.</td>
</tr>
<tr>
<td>Interrelated value sequence (IVS)</td>
<td>The right hand side of the assignment includes the previous value of the variable being defined.</td>
</tr>
<tr>
<td>Single assignment (SAS)</td>
<td>The examined program includes only one assignment statement for the variable.</td>
</tr>
<tr>
<td>Defined sequence (DSE)</td>
<td>The variable has a sequence of values that are statically defined.</td>
</tr>
<tr>
<td>Arbitrary sequence (ASE)</td>
<td>The variable goes through values that are resolved dynamically at run time.</td>
</tr>
<tr>
<td>Singlepass (SPA)</td>
<td>The variable is defined and referred to during a single pass of a loop, and the definition is not referred to in subsequent passes of the loop.</td>
</tr>
<tr>
<td>Abbreviation (ABB)</td>
<td>The definitions used to define the variable reach the use of the definition of the variable.</td>
</tr>
<tr>
<td>Following (FOL)</td>
<td>The variable’s value sequence follows another variable’s or variables’ value sequence(s).</td>
</tr>
<tr>
<td>Partial following (PFO)</td>
<td>The variable’s value sequence may follow another variable’s or variables’ value sequence(s).</td>
</tr>
<tr>
<td>Initial value (IVA)</td>
<td>The definition of a variable is done before a loop, where the variable is redefined.</td>
</tr>
</tbody>
</table>
matching material that were assigned the same role both by the human assigners and the Role Advisor Engine. In Table 2, correctness is reported only for roles that occurred in both materials at least five times. A very small number of teaching examples cannot represent the diversity of that role; neither can a small number of test cases in matching provide reliable results. Such cases are marked with N.A.

It is possible for human role assigners and the RAE to give multiple role suggestions for a variable. In the analysis, the required and non-tolerated FCs were determined so that the RAE suggested only a single role for each variable. In the results of Table 2, an assignment is counted as correct if the role given by the RAE matches any of the roles given by the human role assigners.

The overall accuracy of the RAE was 93%. This can be compared with the accuracy of computer science educators who assigned roles to variables after an introduction to the role concept [11]; their accuracy with short Pascal programs was 85%. The same programs that the educators analyzed were also given to the RAE (using this time both the learning material and the matching material of the above validation as a large learning material). The RAE achieved also 85% correctness with no alternative role suggestions. Thus the reliability of the Role Advisor seems to be comparable to that of computer science educators.

5. CONCLUSIONS

Learning elementary programming can be enhanced by introducing the notion of variable roles to students, but to be effective, role knowledge must also be consolidated. This paper has suggested some ways to consolidate role knowledge, discussed their effects on program creation and program comprehension strategies, and presented a web-based automatic role detection service, the Role Advisor, that can be utilized in teaching and learning. The service uses data automatically learned by machine learning techniques based on role-annotated example programs.

The role detection engine that forms the heart of the Role Advisor was validated using a single cross-validation technique resulting in 93% accuracy. Moreover, the accuracy of the automatic role detection was found to be comparable to that of computer science educators who have been recently introduced to roles.

The current version of the Role Advisor accepts programs in a single programming language. However, as the role detection mechanism is mainly language independent, we plan to continue this work by adding new programming languages to the Role Advisor. We are also working on removing the remaining restrictions and improving the accuracy of the role detection engine.

6. ACKNOWLEDGMENTS

This work was supported by the Academy of Finland under grant number 206574.

7. REFERENCES


<table>
<thead>
<tr>
<th>Role</th>
<th>n_{learn}</th>
<th>n_{match}</th>
<th>Correctness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed value</td>
<td>39</td>
<td>36</td>
<td>97.2</td>
</tr>
<tr>
<td>Stepper</td>
<td>57</td>
<td>34</td>
<td>97.1</td>
</tr>
<tr>
<td>Most-recent holder</td>
<td>28</td>
<td>20</td>
<td>95.0</td>
</tr>
<tr>
<td>Most-wanted holder</td>
<td>7</td>
<td>1</td>
<td>N.A.</td>
</tr>
<tr>
<td>Gatherer</td>
<td>9</td>
<td>12</td>
<td>75.0</td>
</tr>
<tr>
<td>Follower</td>
<td>2</td>
<td>2</td>
<td>N.A.</td>
</tr>
<tr>
<td>Transformation</td>
<td>26</td>
<td>24</td>
<td>91.7</td>
</tr>
<tr>
<td>One-way flag</td>
<td>5</td>
<td>1</td>
<td>N.A.</td>
</tr>
<tr>
<td>Temporary</td>
<td>2</td>
<td>0</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

Table 2: Correctness of role detection