Code Reading Environment according to Visualizing both Variable's Memory Image and Target World's Status

Satoru KOGURE [a*], Ryota FUJIOKA[a], Yasuhiro NOGUCHI[b], Koichi YAMASHITA[c] Tatsuhiro KONISHI[a], & Yukihiro ITOH[d]
[a]*Graduate School of Informatics, Shizuoka University, Japan*
[b]*Faculty of Informatics, Shizuoka University, Japan*
[c]*Faculty of Business Design, Hamamatsu University, Japan*
[d]*Shizuoka University, Japan*
*kogure@inf.shizuoka.ac.jp

Abstract: In this paper, we describe the code reading environment according to visualizing both variable's memory image and target world's status. We had constructed those environment that have two important function. One is the function by which the teacher can reflect his/her intention for instruction on the view of target world's status. Another is the function by which the programming beginner can more deeply understand a program using our proposed environment.

Keywords: Code reading environment

1. Introduction

We think that it is for novice programmers important to understand the relationship between the algorithm and the program in code reading phase observing both memory image of variables in program and status of target worlds. In typically, almost all of novice programmer attend a lecture using educational materials that is included in target world dependency. The beginner can reproduce the status of target world (STW) using the materials. In contract, expert programmer also can analyze a memory image of variables (MIV) in programs step-by-step using typical debugger. In case of simple calculation's task, the representation of STW and MIV are almost same. Although, in case of complex algorithm or data structure, the STW have different representation from MIV. The expert programmer can estimate the STW by observing the MIV using debugger in a program that includes complex algorithm or data structures. Although the beginner programmer cannot because he/she cannot understand the relationship among program, STW and MIV.

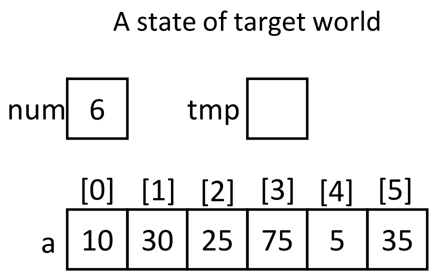
Many systems that support to understand the program and algorithm for novice programmer had constructed by many researcher (Fossati *et al*. (2008), Gabor (2009), Kogure *et al.* (2013), Malmi *et al.* (2004) and Noguchi *et al.* (2010)). We think that these systems have two issues. First issue is that the teacher cannot freely decide the format visualizing the STW. Some systems can reproduce the STW in any step in programs. In those system, although, the teacher cannot reflect his/her intent for instruction to the view of STW. Second issue is that the learner cannot show the STW and MIV synchronously. Almost all system can show the STW for particular algorithm. In the other hand, typical debugger can show the MIV for almost all program that is written for a variety of algorithm. Although, some systems and debugger cannot display the relationship among program, STW and MIV.

The aim of this study is to solve the issues. We had constructed new code reading environment according to visualizing the relationship among a statement of program, STW and MIV for novice learner. This environment has two important function and three basis function. One important function is the function that gives the teacher an environment by which he/she can define the view of STW according to his/her intent. Another is the function that gives the learners an environment by which they can understand the relationship among program, STW and MIV.

2. Fundamental Consideration

2.1 Definition of the State of Target World and the Memory Image of Variables

We think that there are two views for visualizing a behavior of program. One is a state of target world (STW) and another is a memory image of variables (MIV). Figure 1 shows an example of STW and MIV for task of sorting values in array list. As mention in Figure 1, it is easy for novice programming learners to understand a behavior of algorithm using a view of STW instead of a view of MIV. In fact, many systems that support to understand the program and algorithm for novice programmer support a view of STW (Fossati *et al*. (2008), Gabor (2009), Kogure *et al.* (2013), Malmi *et al.* (2004) and Noguchi *et al.* (2010)). In contrast, programming learners must observe a view of MIV to deeply understand a behavior of program or to fix a program that includes a bug.



A memory image of variables

Address	Type	Name	Value
2000	int	tmp	----
2004	int	Num	6
2008	int	a[0]	10
2012	int	a[1]	30
2016	int	a[2]	25
2020	int	a[3]	75
2024	int	a[4]	5
2028	int	a[5]	35

Figure 1. An example of a state of target world and a memory image of variables.

2.2 The Issue that the Teacher cannot Freely Decide the Format Visualizing the STW

Almost all of studies for viewing a STW was for particular algorithm. Some studies' STW don't depend on a particular algorithm but also teacher cannot define a view form of STW. In a case of displaying array list for sorting task, teacher maybe display the array list on horizontal layout (in Figure 1). In contrast, he/she maybe display the array list on vertical layout for stack. Furthermore, the teacher want to change a view of each variable to emphasize the particular variable or to hide some variables. For example, the teacher maybe think that he/she want to hide a view of the variable "tmp" (in Figure 1) for swapping the values of two variable from programming learners. Therefore, we gives the teacher an environment by which he/she can define the view of STW according to his/her intent.

2.3 The Issue that the Learner cannot Show the STW and MIV Synchronously

Almost all of exist studies can show a statement of program and STW synchronously. Typical debugger can show a statement of program and MIV synchronously. Although almost all of exist systems cannot display a statement of program, STW and MIV synchronously. Novice programming learners can easily understand a behavior of program using exist systems that can display a view of STW. Although he/she maybe not understand a behavior of program using debugger that can display a view of MIV. In contrast, an expert programmer maybe understand a behavior of program using debugger. The difference of novice and expert is whether to be able to understand the relationship among a statement of program, STW and MIV or not. Therefore, we gives the learners an environment by which they can understand the relationship among program, STW and MIV.

3. Required Functions for Our Proposed Environment

In order to solve two issues, we define the following five functions for code reading environment:

Func.1. A function that extracts execution history (EH) from a program and generates a program embedded HTML tags for each statement
Func.2. A function that reproduce MIV by using extracted EH
Func.3. A function that give the teacher the environment by which he/she can set the form for the view of the STW for solving first issue.
Func.4. A function that reproduce STW by using MIV and the rule for the view of the STW
Func.5. A function that displays the statement in program, MIV and STW synchronously for solving second issue

There are two types of functions; basis function and important function. Func.1, 2 and 4 are basis functions that we implement using exist methods. Func.3 and 5 are important functions for solving two issues.

3.1 Func.1: A Function that Extracts EH from a Program and Generates a Program embedded HTML Tags for Each Statement

This function is the function that translates a program into two different programs shown in Figure 2. First program includes the code for generation execution history including statement ID and dynamic history ID. The module for this function gives the fragment of statements the unique statement ID when parsing original program. In contrast, the module generates the statement for observing execution state and insert observation statement in original programs. The observation statement dynamically generates history ID when revised program executes. Second program includes the HTML tags for each fragment of statements. The proposed system can display the fragment in program embedded HTML tags and memory image of variable referring execution history synchronously because the statement ID in program embedded HTML tags correspond to the statement ID in execution history.

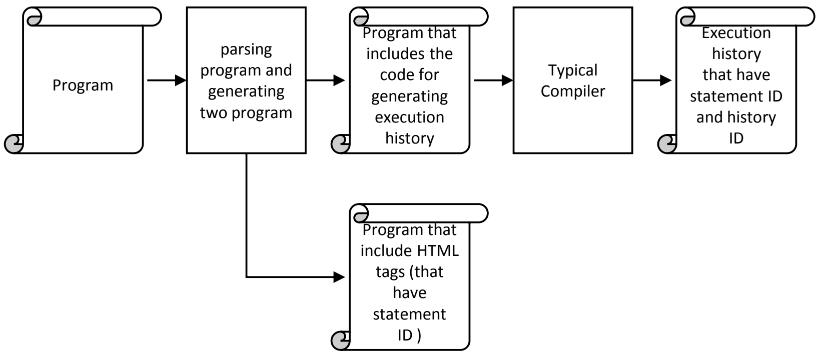

Program
parsing program and generating two program
Program that includes the code for generating execution history
Typical Compiler
Execution history that have statement ID and history ID
Program that include HTML tags (that have statement ID)

Figure 2. Overview of extracting execution history from a program and generating a program embedded HTML tags.

3.2 Func.2: A function that Reproduce a MIV by Using Extracted EH

This function reproduces the state of executing program by referring execution history (Func.2.1) and displays the memory image of variables (Func.2.2). Func.2.1 generates a set of memory image of variables *MIV* by referring *EH* ($EH=\{eh_1, eh_2, \ldots, eh_n\}$) shown in Figure 3.

The certain memory image of variables miv_i includes a set of the four variable information; defined variables' name, types of variables, addresses of variables and values of variables when the system executes from eh_1 to eh_i. EH includes five fundamental operation as follows:

- An operation that allocate memory region for new variable
- An operation that assign new value

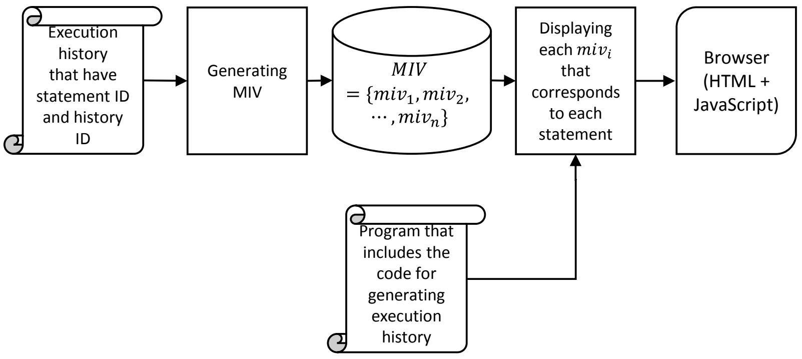

Execution history that have statement ID and history ID
Generating MIV
$MIV = \{miv_1, miv_2, \cdots, miv_n\}$
Displaying each miv_i that corresponds to each statement
Browser (HTML + JavaScript)
Program that includes the code for generating execution history

Figure 3. Overview of reproduce MIV by using extracted EH.

- An operation that allocate memory region for new instance of certain structure (e.g. “struct” in C language, “class” in Java language, etc.)
- An operation that clear memory region for exist variable
- An operation that clear memory region for exist instance

This function is generate the MIV set using the Algorithm 1.

Algorithm 1: generating the MIV

```
1:  $EH \leftarrow \{eh_1, eh_2, \cdots, eh_n\}$ : execution history
2:  $MIV \leftarrow \{\}$ : set of memory image of variables
3:  $cmiv \leftarrow \{\}$ : a memory image of variables after executing from $eh_1$ to $eh_i$
4:  for i=1 to n do
5:      Emulate $eh_i$ on memory image $cmiv$.
6:      Push emulate result to $cmiv$.
7:      $miv_i \leftarrow clone(cmiv)$
8:      push $miv_i$ to $MIV$
9:  end for
```

3.3 Func.3: A Function that Give the Teacher the Environment by which he/she can Set the Form for the View of the STW

First, we define the two types of object that the teacher use for composing a view of STW. First type is a main object that directly correspond to a variable in program. Second type is a sub object for explaining a role of a main object, explaining a relationship between a main object and other main object, or setting an array/table layout of each object. The teacher can specify each attribute value (e.g. object’s position, size, color, and so on) of each object. Therefore, the teacher can set a timing in which an object creates, deletes, shows and hides by using statement ID or history ID.

The teacher prepares the configuration file for a view of STW described above paragraph. Table 1 shows the content of configuration that teacher describes for main/sub objects and timing of creating/deleting/displaying/hiding each object.

Table 1: Types of object and Attributes for Configuration of STW.

	Operations	Types	Common Attributes
Main Object	Create, Delete Update	Circle, Square Rectangle	Corresponded variable, Position, Width, Height, Color, Line weight Line style
Sub Object	Create, Delete Update	Connecter, Table Label, Line, Balloon	Target main object ID, Position, Width, Height, Color, Line weight Line style

The teacher can use three types of actions described in Table 1. "Create" action of main object is an action when observing an execution history in which a new variable is created. "Delete" action of main object is an action when observing an execution history in which an existed variable is cleared. "Update" action is an action for changing any attribute (e.g. color for emphasizing) at any timing that the teacher decides using statement ID or history ID).

The teacher can also use sub object in Table 1 for explaining a role of a main object or a relationship between a main object and other main object. For example, "Connecter" type is a connecter that connect a main object to other main object. The teacher can present the reference relation using attribute "Line style" (e.g. arrow). "Balloon" type is a balloon help for a main object.

The a rule description of rule set for a view of STW includes five items; "condition", "operation", "object name", and "set of attribute-value". "Condition" item is for specifying a timing of adapting the rule. The teacher can use six comparison operation; "==", "!=", ">=", "<=", ">" and "<" and three types of operands; "immediate number", "variable in program" and "statement or history ID".

3.4 Func.4: A Function that Reproduce STW by using MIV and the Configuration for the View of the STW Created by the Teacher

The system reproduce certain stw_i according to miv_i and configuration rule set for the view of the STW created by the teacher. The system find rules in which a comparison operation is satisfied. If the system find match rule, the system executes this rule. If this rule is "create" operation's rule for main object, the system retrieve a value of variable corresponding the rule from miv_i and display a main object according to object name, variable value and a set of attribute-value.

3.5 Func.5: A Function that Displays the Statement in Program, MIV and STW synchronously

Learners can show a reproduced execution process of program code using our proposed environment. Our proposed system have execution history, EH, as common knowledge that each function use for executing own affair. Learners can use "next" or "prev" command for moving focused execution history eh_i to next execution history eh_{i+1} or previous execution history eh_{i-1}. Therefore, the system reproduces miv_{i+1} or miv_{i-1} for memory image of variables and also reproduces stw_{i+1} or stw_{i-1} for a state of target world.

4. Implementation

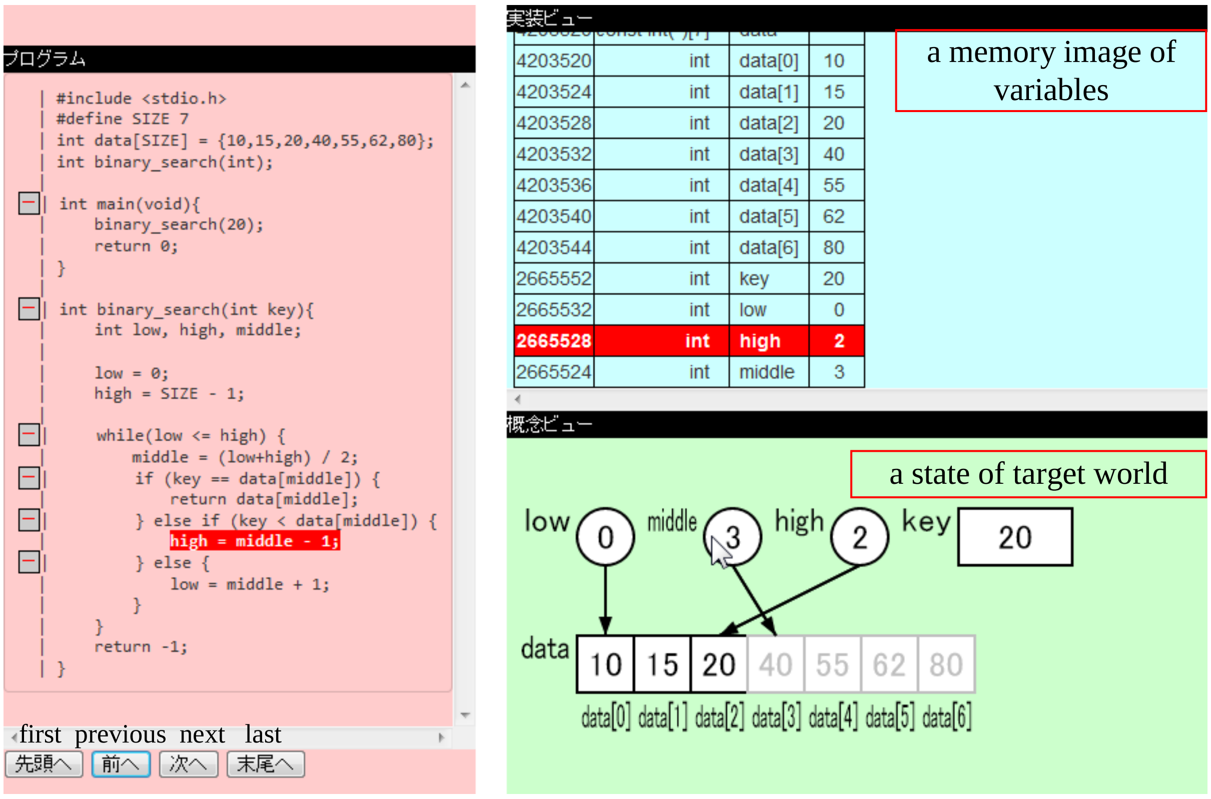
Figure 4 shows an overview of our proposed environment. First, the left side area is for displaying program. Second, the right/top side area is for displaying a memory image of variables. Last, the right/bottom side area is for displaying a state of target worlds. In this study, target programming language is C language. We implement the parser of C language using Parse::RecDescent (Perl module for generate recursive-descent parser, referring http://search.cpan.org/~jtbraun/) and we implement the other module using HTML and JavaScript. We use JSON format as the representation of execution history.

For primary evaluation, we monitored what minutes the one of author spends creating the rule set for a view of STW. We select three situation; binary search task, linked list task, and other linked list task. As a result of evaluation, we obtained 36, 48 and 37 minutes, respectively. Therefore, we suggest that the teacher and learners maybe can use our environment in real classroom lecture.

5. Conclusion

We had constructed new code reading environment according to visualizing the relationship among a statement of program, STW and MIV for novice learner. This environment has two important function. One important function is the function that gives the teacher an environment by which he/she can define the view of STW according to his/her intent. Another is the function that gives the learners an environment by which they can understand the relationship among program, STW and MIV. We easily evaluate first important function. As a result, we suggest that the teacher and learners maybe can use our environment in real classroom lecture.

In the future, we will refine the second important functions. Therefore, we will evaluate the advantage of our proposed two function on real classroom lectures.



```
#include <stdio.h>
#define SIZE 7
int data[SIZE] = {10,15,20,40,55,62,80};
int binary_search(int);

int main(void){
    binary_search(20);
    return 0;
}

int binary_search(int key){
    int low, high, middle;

    low = 0;
    high = SIZE - 1;

    while(low <= high) {
        middle = (low+high) / 2;
        if (key == data[middle]) {
            return data[middle];
        } else if (key < data[middle]) {
            high = middle - 1;
        } else {
            low = middle + 1;
        }
    }
    return -1;
}
```

[illegible]	const int( )[7]	data	
4203520	int	data[0]	10
4203524	int	data[1]	15
4203528	int	data[2]	20
4203532	int	data[3]	40
4203536	int	data[4]	55
4203540	int	data[5]	62
4203544	int	data[6]	80
2665552	int	key	20
2665532	int	low	0
2665528	int	high	2
2665524	int	middle	3

low	middle	high	key
0	3	2	20

data[0]	data[1]	data[2]	data[3]	data[4]	data[5]	data[6]
10	15	20	40	55	62	80

Figure 4. Overview of our proposed environment.

Acknowledgements

This study was supported by Japanese Grant-in-Aid for Scientific Research (B) 24300282.

References

Fossati, D., Eugenio, B. D., Brown, C., & Ohlsson, S. (2008). Learning linked lists: experiments with the iList system, *Proceedings of the 9th International Conference on Intelligent Tutoring Systems*, 80-89.

Gabor, T., (2009). Algorithm visualization in programming education,. *Journal of Applied Multimedia*. *4*(3), 68-80.

Kogure, S., Okamoto, M., Yamashita. K., Noguchi. Y., Konishi, T., & Itoh, Y. (2013). Adapting guidance and externalization support features to program and algorithm learning support environment. *Proc. of the 21st International Conference of Computers in Education*, 418-424.

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppala, O. & Silvasti, P. (2004). Visual algorithm simulation exercise system with automatic assessment: TRAKLA2, *Informatics in Education*, *3*(2), 267-288.

Noguchi, Y., Nakahara, T., Kogure, S., Konishi, T., & Itoh, Y. (2010). Construction of a learning environment for algorithm and programming where learners operate objects in a domain world', *International Journal of Knowledge and Web Intelligence*, *1*(3), 273-288.