

Extension of the educational system that can generate explanation of programs

- For handling programs in which pointers are used -

Tatsuhiro Konishi, Toshikazu Kawai, Satoru Kogure, Yukihiro Itoh,,
Faculty of Information, Shizuoka University, Japan

Abstract: We have developed an educational system that helps novice programming learners. Our system can understand meaning of programs by simulation in domain world models and generate explanations that show functions of the programs using vocabularies of domain words. Our conventional system cannot handle programs in which pointers are used. Therefore we extend the system for understanding complex programs that contain pointers and for generating explanation of such programs.

Keywords: Intelligent Educational System, Programming education, Simulation, Program understanding

Introduction

Educational systems for programming can be classified roughly into two types; systems for diagnoses and systems for explaining / visualizing. The former has ability to find bugs in learners' programs. The latter shows learners behavior of programs as explanation text or graphics. Zeus[1], TANGO[2], and ANIM[3] can generate animation illustrating behavior of a program. However, they cannot generate the result directly from source codes, but they need additional specification such as commands for drawing. Therefore it isn't easy for learners to use such systems. It is desirable that such systems can understand target programs without additional specification and can explain / visualize the program. Tracers for debugging can generate behavior of programs directly from the source codes, but tracers use only vocabularies of programming language. It is more desirable that such systems use vocabularies including concepts in algorithm (for examples, abstract data structures such as "linked lists" and concepts in the domain world of the problem such as "a sorted list").

We have developed an educational system that has ability to understand programs written in C language, and to generate explanations of the programs using vocabularies of abstract data structures and domain worlds[4-7]. In order to understand the target programs, the system reproduces the behavior of the programs on three types of models: the data structure model, the abstract data structure model and the domain world model. The system observes the reproduced behavior to understand meaning of target programs. Then it generates explanation of the target programs using vocabularies including abstract data structures and concepts in the domain word. Our previous system couldn't cope with pointers and structures. So we redesign our data structure model and understanding method to be able to handle them. In this paper, we propose the extended methods, and report on the extended system.

1. Our conventional system

Our system support learners by two methods: (a) for understanding algorithm: Learners input a sample program. The system gives them explanation that has hierarchical structure. Learners can select grain-size of explanations: the smallest size corresponds to a statement, and the larger size corresponds to a block. It helps learners to understand hierarchical structure of the algorithm. When learners put a mouse cursor on an explanation or program, the corresponding part of the program or explanation turns red. By this function, they can learn correspondence between each part of the algorithm and of the input program. (b) for debugging learner's program: Learners input their program. If the program has any bugs, the explanation output from the system will be different from their expectations. So they can find their program is wrong by themselves.

Fig.1 shows the structure of the system. The system has three world models: the domain world model, the abstract data structure model and the data structure model. The former two models are called "higher models". The domain world model represents concepts in the domain world. In this paper, the domain world means the world that programmers imagine when they design problem solving processes. For example, sorting problems has the domain world including data lists each of which has "size" and they are comparable by the size. The abstract data structure model represents abstract data structures such as "lists", "stacks", etc. The data structure model represents data structures defined in a programming language, such as variables, arrays, etc. The system has a set of pattern recognizing programs called "observers". The observers always observe the world models. As soon as they find a certain pattern, they write the result on one of the world models. Thus observers recognize correspondences between concepts or operations in a world model and another.

The system simulates programs in the following way: At first, the system analyzes the input program and simulates changes in the data structure model. Then, observers observe the data structure model. If they find some patterns that can be regarded as a state in a higher model, the system reproduces the state on the higher model. Next, observers observe the abstract data structure model. Then the system reproduces the state of the domain world model by the same method. As a result, all world models reflect changes caused by the input program.

In order to generate explanation of a statement or a block, the system analyzes the difference of the states before and after the execution of the statement(s) in each world model. For example, if a data list is arranged in random order, and the list is sorted after executing a block, then the explanation for the block is "sorting [datalist]". The system has templates for each meaningful pattern of such differences in order to generate texts. When the system cannot find meaningful differences, it uses templates defined for each function of C language and each type of block structure; sequential structure, iterative structure, and selective structure.

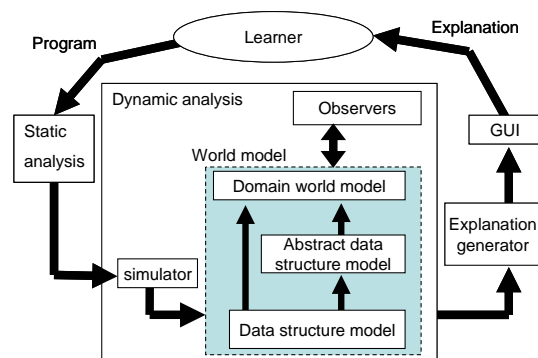


Fig.1 Structure of the system

2. Methods for constructing extended system

The conventional system was designed for handling only simple variables and 1-dimension arrays. Therefore, it can support only novice class of programming. So we have to extend our system to handle more complicated data structures constructed using pointers, structures and N-dimension arrays, such as linked lists or trees. In order to handle them, we extend the data structure model, the simulator, and program understanding method. We represent data structures by frames in the data structure model. We add frames for pointers, structures and N-dimension arrays. We also extend the simulator for processing of address data, such as &-operator, *-operator, ++(or --)-operator with address value, assignment of address value, and functions called by reference.

We consider whether the conventional observers are applicable to understand meanings of pointers. Conventional observers can recognize that the pointer is pointing to an entity in the higher model (Ex. “This pointer is pointing to a node in a linked list” in the abstract data structure model). However, such recognition is not enough to generate good explanation. By case study on standard textbooks of programming, we find that explanations for pointers usually represent the role of pointer in the algorithm. For example, “pointer-x has a role pointing to the current focused node” in algorithm searching a node in a linked list, etc. As a result, the following types of recognition should be generated on meanings of pointers.

- (a) What kind of entity in the higher models corresponds to variable that is pointed by the pointer.
- (b) What kind of role the pointer has in the problem solving process of the program.

Our method to generate these recognitions is as follows:

Recognition (a): Identify meanings of the variables pointed by pointers by conventional observers.

Recognition (b): It is necessary to observe transition of pointed entities by the pointer. For example, suppose a program for removing a node satisfying a condition from a linked list. At first a pointer points to the root node, then it changes pointing node to the next node one after another. In such a case, the pointer must have a role to point to a node that is currently focused. In order to recognize such a role, our extended system stores transition of the nodes pointed by each pointer, and observes transition patterns. So, we add new type of observers observing transition pattern of states of pointers.

3. Example of program understanding

We show an example of program understanding by the extended system. In this section, we call the abstract data structure model “A-model”, and the data structure model “D-model”. The target program is inserting a node to a linked list (Fig. 2). We don’t use the domain world model because explanation of the program is represented by only vocabularies of A-model and the D-model. A-model has to have the following entities and operations in order to handle linked lists. [Entities: Nodes, Pointer sticks, Data, Lists][Operations: Insertion, Search, Swap] And the following observers are necessary for understanding meaning of a group of operations from reproduced actions in A-model: [R1: Recognizing that a node is inserted when a new node is set as the next node of the focused node] [R2: Recognizing that process of searching node is done when an iterative block in which focus is moving from a node to the next node one after another is finished].

In order to understand meanings of operations, the system has to recognize the following roles or statuses of entities in A-model. The system recognizes them by observers. We show some examples of conditions that make observers recognize the roles or statuses in [].

- Nodes: a) This node is the Nth node in the list [by analyzing position of the each node in the list, after a node is inserted]. b) This node is newly generated.
- Pointer sticks: c) This stick is pointing to a node. d) This stick is pointing to NULL
 - e) This stick is newly prepared [when a pointer is generated in D-model.]
 - f) This stick is pointing to a node to be focused [when a node pointed by a stick has been moved from first node to the following nodes one by one for each iterative execution of a loop].
 - g) This stick is pointing to a node to be inserted.
- Data: h) This datum is a value stored in a node.
 - i) This datum indicates position to insert a node [when insertion of a node is recognized, and a datum equals to position number of the node].
- Lists: j) This list has N pieces of nodes.
 - k) This list is initialized. l) This list is rewritten.

```

void insert ( int n ,int data) {
  struct CELL *x, *p;           <1>
  x = &header;                 <2>
  for(i=0; i<n; i++){          <3>
    if (x==NULL){             <4>
      break;                   <5>
    }
    x=x->next;                 <6>
  }
  p=malloc(sizeof(structCELL )); <7>
  p->value = data;             <8>
  p->next = x->next;           <9>
  x->next = p;                 <10>
}

```

Let’s trace process of understanding the

Fig.2 Sample program

program in Fig.2. Suppose the program is called by the statement “insert(2,3)” (A node having data “3” is inserted to the second position of the list). In the following, we represent the Nth iterative execution of a loop “Statement<*>-n”.

The initial condition(Fig.3): Before the target program is called, global variable “header” has been declared in the main function. The role of “header” has been recognized as “the root node”. From the root node, 2 nodes have been linked.

Statement<1>: Pointer frames x and p are generated in D-model. By observer-e), stick-x and p are generated in A-model. Because of deference between status before and after <1>, the meaning of <1> is identified as “Prepare pointer stick-x and p”.

Statement <2>: Pointer “x” points to the variable “header” in D-model. The meaning of <2> is “Let stick-x point to the root”.

Statement<3>-1: Condition “i<n” is evaluated. i=0, n=2. So the system executes the body of the loop.

Statement<4>-1: Condition “x == NULL” is evaluated. It is false.

Statement<6>-1: Pointer-x changes its pointing target to the next node of the current node in D-model. In A-model, stick-x changes its pointing target to the “0th” node. The meaning of <6>-1 is identified as “Let stick-x point to the 0th node”. Previous state of stick-x is stored as history data.

Statement<3>-2- <6>-2: The understanding processes are similar to the statement <3>-1- <6>-1.

Statement<3>-3: “i<n” is evaluated. i=2,n=2. So it exits the loop. Then observer-f) observes the transition pattern data. It reports that “Stick-x has a role to point to a node to be focused”. Meaning of this iterative block is identified as “a block searching a node having a certain property” by the observer R2. Fig.4 illustrates the world models at this moment.

Statement<7>: Simulating the malloc function, the system generates a structure frame in D-model, and a node in A-model. Then, the role of stick-p is identified as “pointing to a node to be inserted”. As a result, the meaning of <7> is identified as “Let stick-p point to the node to be inserted”.

Statement<8>: Value of variable “data” is assigned to the structure frame pointed by pointer “p” in D-model. The meaning of <8> is identified as “Set the value of data on the node to be inserted”.

Statement<9>: The next pointer of the node pointed by “p” points to the next node of the node pointed by “x” in D-model. The meaning of <9> is identified as “Set the next node of the focused node as the next node of the node to be inserted”.

Statement<10>: It is reproduced in A-model that “The node to be inserted is set as the next node of the focused node”. It is recognized that a node is inserted to the linked list by the observer R1.

When some nodes are inserted in a linked list, observer-a) and j) changes statuses of nodes and the list respectively. As a result, inserted list is recognized as “2nd” node in the list. It is also recognized by observer-i) that the variable “n” indicates position to insert a node in a list. Fig.5 illustrates the world models at this moment.

The system generates explanation text from the recognized meanings of statements and roles of variables by using templates. Templates are prepared for each operation or observer. For examples:

- Observer-e) (a stick is prepared): “Prepare a stick-[variable name] for [Recognized role of the stick]”
- Operation of assigning a value to a new node: “Set [Recognized role of the value] to the new node”
- Observer R1: “Insert a node: Position is [Recognized position of insertion], datum is [datum stored in the inserted node].”

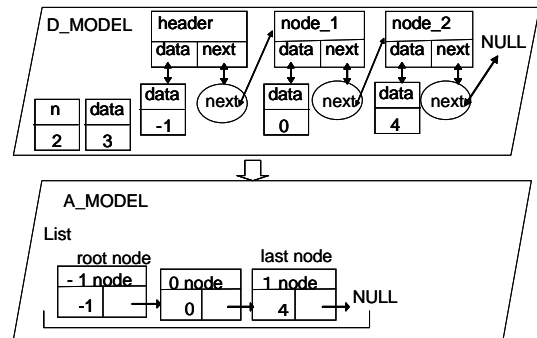


Fig.3 The initial state of the world models

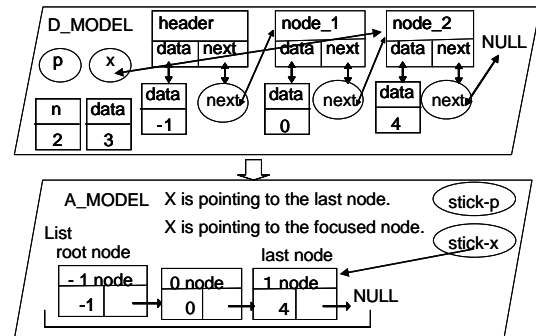


Fig.4 World models after exiting the loop

4. Implementation of the extended system

We constructed the extended system based on methods mentioned above. The system is implemented in Common Lisp and Tcl/Tk. Currently, the system has observers and templates necessary for handling insertion and deletion to/from a linked list. Fig.6 shows output of the system. Original explanation is written in Japanese and it is translated. The target program is the sample program in Fig.2 and main function calling it. The text is generated based on the result of understanding process mentioned in 3. All messages work as a kind of button. At first, only a button “Start explanation” is displayed. When learners click the button, smaller grain-sized explanations are displayed.

5. Conclusion

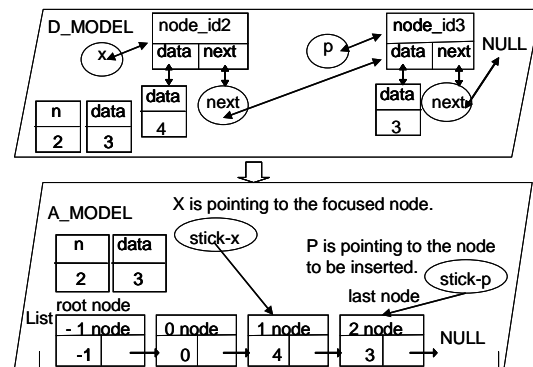
Currently, we are developing additional observers necessary for handling other types of abstract data structures than linked lists. We think our methods are applicable to almost of other types of abstract data structures. We have evaluated the conventional system by experiment and confirmed effectiveness[4]. We are planning to apply the extended system to actual learners in order to evaluate its effectiveness.

Acknowledgements

This research was partially supported by the Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (B), 17300265, 2005-2007, and Grant-in-Aid for Scientific Research (B),20300267, 2008.

References

- [1] Brown, M.H. (1992). Zeus:A System for Algorithm Animation and Multi-View Editing. *Computer Graphics*, 18(3), 177- 186.
- [2] Stasko, J.T. (1992). Tango:A Framework and System for Algorithm Animation. *IEEE Computer*, Vol.23, No.9, 27-39.
- [3] Bentley, L.B., & Kernighan, B.W.(1991). A System for Algorithm Animation Tutorial and User Manual. (*Computer Science Technical Report No.132*), AT&T Bell Laboratories.
- [4] Anma, F., Konishi, T., & Itoh, Y. (2004). Construction and Evaluation of an Educational System that Explains the Domain-oriented -explanation of Program's Behaviors. *Proc. of ICCE 2004*, 839-845.
- [5] Ando, T, Kawada, Y, Itoh, R, Konishi, T. & Y.Itoh. (2000). An Educational System that can Visualize Behaviour of Programs on the Domain World. *Proc. of ICCE 2000*, 1087-1095.
- [6] Itoh, R., Nishizawa, M., Konishi, T., & Itoh, Y. (1998). On Constructing an Educational System that Analyzes Buggy Programs and Makes Domain Oriented Explanations. *Proc. of ICCE'98, Vol.1*, 432-440.
- [7] Itoh, R., Nishizawa, M., Konishi, T., & Itoh, Y. (1997). On Analysis of Novice Pascal Program Based on Simulation to Extract Information for Education. *Proc. of ICCE'97*, 485-492.



In A-model, “n” is the position to insert a node in a list, and “data” is the value stored in a node.

Fig.5 World models after insertion

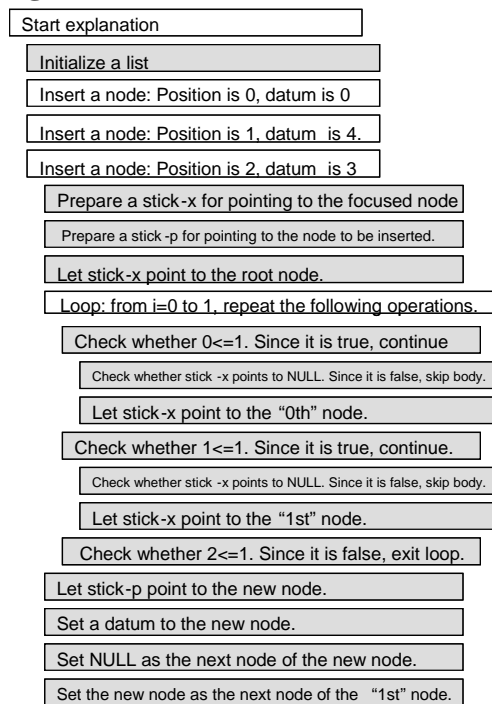


Fig.6 Output of the extended system