Title	動的解析に適したインタプリタに関する研究
Author(s)	蔡，遠航
Citation	
Issue Date	2001-03
Type	Thesis or Dissertation
Text version	none
URL	http://hdl.handle.net/10119/1427
Rights	
Description	Supervisor:片山 卓也 教授，情報科学研究科，修士

Research On A Interpreter Suitable For Dynamic analysis

Enkou Sai

School of Information Science,
Japan Advanced Institute of Science and Technology

February 15, 2001

Keywords: dynamic analysis, interpreter, execution information, Sapid, CASE tool platform.

There are static analysis and dynamic analysis as analysis of a source program. In development and maintenance of software, debugging, a test, etc. of a program have an important role and dynamic analysis is confirmed at these. Dynamic analysis means giving a certain input to a source program, performing it, analyzing it using the information on the program state at the arbitrary execution step (it is called execution information). Dynamic analysis can perform more advanced analysis with much information compared with static analysis. For example, there are important things in the error of the software that is hard to find in static analysis, such as an error about the value of a pointer. These errors become possible discovering in dynamic analysis.

However, it is often more difficult to implement dynamic analyzers than to implement static ones, since dynamic analyzers have to treat dynamic information such like stacks, heaps, registers, in addition to abstract syntax trees and symbol tables that static analyzers treat. In the existing dynamic analyzers, the function of abstracting and modeling this dynamic information is inadequate. The information is in a characteristic format the dynamic analyzer has, and there is no structure to offer the information outside. Therefore, the reusability of this information is low. The interpreter for source programs is implemented respectively for every dynamic analysis tool. Although implementing the underlying interpreter is not essential when implementing dynamic analyzers, it requires much cost.

If the execution information on the program is modeled, abstracted and then offered, it will become much easier to implement dynamic analyzers. In order to store execution information on a program, a general way is performing the program using interpretation machines, such as a compiler or an interpreter, and storing the execution information simultaneously with the interpretation machines.

If a program is performed by using a compiler or an interpreter, it will perform on the basis of dynamic information in original binary format. In this case, interpreting the dynamic information in original binary format using the format that is different for every system, will be extra work. To eliminate this extra work, dynamic information should be modeled, abstracted, and made easy to use in dynamic analysis. However, there are few interpreters that offer such dynamic information in the present condition.

On the other hand, Sapid is proposed as a fine-grained repository for the purpose of becoming the platform of CASE tools. This is a software repository that is made based on a model called I-model. I-model is an E-R model that describes software on the basis of the structure of software, and is the data structure that specialized in the C language and modeled the static information such as the information on an abstract syntax tree and a symbol table.

The Repository based on I-model is fine-grained compared with a repository of module units or that of function units. To the result analyzed by static analysis of the source code based on this repository, the virtual machine that can direct interpret and execute it can be created easier. This shows that Sapid is effective in dynamic analysis.

Sapid is provided with an interpreter named Sint of the C language. However, as for Sint, it is insufficient to offer the dynamic information at the steps of execution of a program. It is because the modeled and abstracted execution information is not offered.

Therefore, Sapid is excellent in offering static information but it is still inadequate for offering dynamic information. If program execution information is becoming storable by using Sint, when developing a CASE tool, we can offer analyzers that are excellent in both of static and dynamic analysis functions, which are often required. We can also expect labor-saving of development, data integration among CASE tools.

Thus, in case proposing and implementing an interpretation machine of the program which is suitable for dynamic analysis, extending Sint rather than making a new interpretation machine from the scratch. This can be performed labor-saving of implementation, and it is suitable to the essence of Sapid as a CASE tool platform.

In this research, the model and APIs about execution information for dynamic analysis were proposed. Based on it, Sint was extended to be an interpreter suitable for dynamic analysis.

In realization of this, first, to make the interpreter suitable for dynamic analysis, we took slicing and debugging for the examples. By these examples we considered the demand matter on dynamic analysis and the advantage of treating execution information logically. The demand matter is ”the information about the value of the variable, an address, the state of a stack, and the performed statement at each execution step is acquirable”. To model execution information, we bore ”offer a model also with high flexibility and high degree of abstraction” in mind and discussed the relation between the degree of abstraction and flexibility.

Base on these execution information was sorted out. We paid attention to next six kinds of information: the execution step, the performed statement, the variable, and the value of the variable, the area, and the state of the stack showing a function call relation. We proposed a model about this execution information. Moreover, we extended both Sint and SIP2, a library supports the dynamic analysis. By this extension, we made Sint can extract and store the above-mentioned six kinds of execution information.The execution information model consists of six classes and six kinds of relation, and we offered nine APIs together with the model. By discussing the theoretical realization technique of a tool investigating a function call relation and a variable history tool, this interpreter having a certain validity to realize dynamic analysis tools was shown.

By realization of the interpreter proposed and implemented by this research, the logical concept such as a variable or a stack can be directly treated now. Moreover, when developing a CASE tool, we can offer analyzers that are excellent in both of static and dynamic analysis functions, which are often required. We can also expect labor-saving of development, data integration among CASE tools.