Dynamic Slicer Implementation with Sapid

Sasai Takanori

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

February 15, 2000

Keywords: dynamic slicing, pointer analysis, Sapid, CASE tool platform.

1 Background

Dynamic analysis is very important to develop and maintain programs. To debug dynamic errors like memory leaks, we need to know dynamic information of the programs. To mplement dynamic analyzer, we often need to reconstruct the whole execution environment including lexcal-analyzer, parser, static semantic analyzer and so on. This makes it difficult to implement dynamic analyzer, although they are not essential. Therefore there are very few implementation of practical dynamic analyzer.

Program slicing techniques, proposed by Weiser, is common solution of some problems in software engineering for debugging, testing, reuse and so on. Slicing is a technique to extract parts of programs by tracing control and data flow related some data items. There are two kinds of slicing: static slicing and dynamic slicing. Restricting inputs, dynamic slicing can make more precise slice of program than static slicing. Moreover algorithm of dynamic slicing is not so complicated. Therefore many slicing algorithms have already been proposed, but there are few practical slicing tools. It is because dynamic slicing tool dynamically analyze the program, and implementing dynamic analyzer is difficult as mentioned above.

On the other hand, fine-grained software repository Sapid is proposed and implemented by Nagoya Univ. Sapid offers the software database called SDB and APIs to access SDB. SDB is based on software model called I-model. I-model is a ER-model consists of 12 entities and 29 relations. Each entity almost corresponds to a symbol in the grammar of C language, and each relation corresponds to the relation between syntax. I-model is fine-grained enough to realize C interpreter based on the data stored in the software repository. Therefore Sapid is expected to be powerful tool for dynamic analysis.

2 Purpose and Approach

The purpose of this paper is to implement dynamic slicer for C language using Sapid experimentally, and to evaluate Sapid applyied to a dynamic analysis. And we research factors which make dynamic slicing difficult.

There are three difficult factors to implement dynamic slicer for C language. First it is difficult to implement execution environment. We use Sapid to resolve this problem. There is API called SIP2 in the Sapid which supports dynamic analysis by abstract executing the programs.

Second, it is difficult to cope with pointers, structures and type casting in C language. C language allows very free memory access using pointers, and it causes pointers to refer to invalid addresses. The memory layout of structures is implementation dependent. So casting structures often cause uncertain field access. For this problem we use approximation methods Points-to set analysis which are well-known techniques to deal with in pointer analysis. But this problem is too difficult, we made assumption that 1)there is no type cast except for return value of malloc(), 2)there is no operation with side effects in the formula.

Third problem is library functions and system calls. We have no way to know what variables are refered to and defined in the function calls without source code. Therefore we can not trace data flow in such cases. So, we made assumption again, 3)there is no function calls or system calls except for malloc() and free().

3 Conclusion

Pointer analysis is very big problem for dynamic slicing. C language allows to access invalid memory address freely, and it is very difficult to analyze. Since C language has many implementation dependent features, it is difficult to implement portable analyzer. But it is not preferable to use pointer which refers to invalid memory address, or to code implementation dependent programs. And there is no need to code such a dirty program, unless which is system programming.

We use Sapid in this research. We confirm Sapid is effective to reduce process of coding dynamic analyzer. C language interpreter sint, which use Sapid, has 104 lines in its source code. There is also C language interpreter CINT, which does not use Sapid, has 85,041 lines in its main source code. CINT supports more features than sint, it is not fair comparison, but size of two souce code is too differ.