Code Level Context Sensitivity Exploration Algorithm

Yan Lijing[1,2,a] , Shan Zheng[1,2,b] , Xu Xiaoyan[1] , Xue Fei[1]

[1]The State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450002, China

[2] Information Engineering University, Zhengzhou 450002, China

[a]jing894019143@163.com, [b]zzzhengming@163.com

Keywords: Behavior model; Function call graph; Context sensitivity; Source code level; Inter-procedure analysis

Abstract. The context sensitivity is an important property in program analysis which can improve the analysis accuracy of the function call context of the source codes and helps to improve the abilities of the compiler optimization and verification procedures. The automata model based on the function call of the program source code is an effective method of modeling the behaviors of the current software, but this method does not consider the context sensitivity of the software function calls. To solve this problem, on the base of high-performance open source compiler Open64, we achieve a source code level context sensitivity exploration algorithm to analyze the relationships between the points of the function calls. Using the iterator, by the circulating way, the algorithm traverses the calling relationships between functions and obtains the context sensitive information analysis results. With this method, it can guide the source code modeling of using the single context sensitivity or the mixed context sensitivity models. In the benchmark sets SPEC2006 and NPB3.3.1, we verify the correctness of the algorithm, and the results show that the proposed algorithm can detect the context sensitivity of the source code level, and the hybrid context sensitivity modeling approach which we used, comparing with other modeling methods, has had a higher accuracy.

1. Introduction

Software behavior modeling main purpose is to get the statusinformation of the softwares, so as to analyze the behavior of the software. In order to efficiently obtain the state information of software, we need to design a reasonable modeling method.The researchers have proposed many methods of modeling software behavior: the short sequence model based on system calls, the automaton model, the data flow model and so on. Among them, the automata model gets the function calls' relationships within the range of all functions, typically models include the non-deterministic finite state automata (NFA), pushdown automata (PDA) and Dyck model [1]and so on. These modeling methods are related to the context sensitive models.

However, using a automata to model the behavior does not take into account software's context sensitivity. If the called program context sensitivity does not exist, utilizing context sensitive automaton model will bring redundant computation overhead. If the program exists the called context sensitivities in many places, using context insensitive analysis method, existing the impossible path problems,it will lead to the software attacked at runtime. In order to ensure the model calculation precision and improve the scalability, it is necessary in front of the source code modeling to analyze in the program function context sensitivity.

Based on software behavior modeling for application requirements, we put forward a software source code context sensitive detection algorithm before softwaer behavior modeling. The algorithm is mainly aimed at the statistical information of the function invocation context sensitivity. According to the acquired context sensitive information to guide behavior modeling software:(1)The source code which is function context insensitive can build FSA model;(2)Context sensitive call points in the source code can build software model using PDA, Dyck model;(3)Functions in the source code contain both the context sensitive and insensitive call points program, can take the combination of FSA with PDA modeling software in the form of behavior.

2. Related research

In recent years, Scholars put forward a series of behavior models associated with program context analysis. Context insensitive software behavior models include finite state automaton[1], abstract stack model[2] and the call graph model[3],and so on.Finite state automata (FSA) to establish behavior models by statically analyzing software source codes, does not extract system call context information. Abstract stack based on the static control flow analysis of program source code, builds call graph model first, and then obtains the abstract stack model through the call graph model,is essentially a context insensitive non-deterministic FSA. And call graph model with the function calls actually is an NFA, also does not have the context sensitivity.

Context sensitive software behavior models mainly include traversing the call stack,the call context tree, Dyck model and the HFA model, etc.Call stack [4] as the most convenient way to get the context information, which uses a virtual stack structure as the research object.Spivey[5] proposed dynamically creating a call context tree, by monitoring the procedure's position in the calling context tree to make credible evaluation. A probabilistic model of the calling context proposed by Bond et al[6]had a higher efficiency than calling context tree.Dyck model[7]can delete redundant invalid calls and sub-paths without system calls by using the method of dynamic compression, is concerned the application context sensitivity of binary code; HFA[8] is essentially a DPDA model, not only can obtain better precision, but also is a context sensitive model.

These studies in front of the software model didn't consider the context sensitivity analysis as an important role on the accuracy and efficiency of the improvement process. Context sensitive detection algorithm by analyzing the relationships between the call points context sensitivity, will be more beneficial to choose the right model for behavioral model.

3. Interprocedural analysis

The realization of the source code level context sensitivity exploration algorithm mainly depends on Open64 compiler [9] generated function call graph.This paper mainly elaborate from Open64 compiler analysis framework,extended function call graph and call graph generation and traverse. Function call graph generated during the interprocedural analysis,the framework is shown in figure 1.

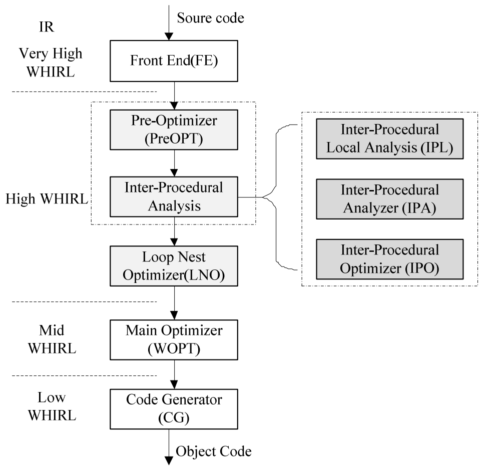


Fig.1 Open64 Inter-procedure analysis frame

3.1 Interprocedural call graph generating formalization description

Function call graph generation mainly depends on the source code, in this paper, the formalization description of function call graph generation process: Setting the input source $S_{sourcecode}$, the call graph generation rely on interprocedural analysis[10](expressed as ψ),mainly include: interprocedural local analysis, interprocedural analysis, interprocedural optimization. The latter analysis among the three analysis processes depend on the result of the former analysis,we set up the three analysis processes for the S_{IPL},S_{IPA},S_{IPO}.Through the above analysis processes,we obtain the source code for the function call graph, set to $S_{callgraph}$, having the following formula:

$$S_{callgraph} = S_{sourcecode}\psi(S_{IPL} \to S_{IPA} \to S_{IPO}) \tag{1}$$

Among them, the process of local analysis (S_{IPL}) main function is to collect all the summary information, we put these aggregate information generically indicated as *i*,then S_{IPL} can be expressed as:

$$S_{IPL} = \sum_{n}^{0} i \tag{2}$$

In the above formula, *n* represents the total number of required summary information.

Interprocedural analysis (S_{IPA}) is establishing the call graph using the summary of information obtained.during the S_{IPL}. including interprocedural constant propagation, inline replacement and alias analysis and other analysis methods. There are no absolute relationships between these methods. The relationships between the analysis methods are represented as |. Set these analysis methods respectively, M, G, P... , S_{IPA} can be expressed as:

$$S_{IPA} = M(S_{IPL}) \mid G(S_{IPL}) \mid P(S_{IPL})... \tag{3}$$

Interprocedural optimization (S_{IPO}) mainly based on the results of S_{IPA}, obtains intermediate representation of the source code, converts into a function call graph. We set up the transformation process is $Change$, the S_{IPO} can be expressed as:

$$S_{IPO} = Change(S_{IPA}) \tag{4}$$

Through the above formalization, formula (1) can be described as:

$$S_{callgraph} = S_{sourcecode}\psi(Change(M(\sum_{n}^{0} i) \mid G(\sum_{n}^{0} i) \mid P(\sum_{n}^{0} i)...)) \tag{5}$$

3.2 extended function call graph analysis

Program call graph[11] is used to indicate program interprocedural call relationships. Every call point usually needs to be represented by one single node in the figure. In this case, the call graph is actually a multi-figure.The accuracy of the call graph is a direct impact on the accuracy of the function call context sensitivity analysis. Extended call graph is an improvement on the traditional call graph. Each node in the figure represents a function. Extended call graph compared with the traditional call graph extends some additional information. For the same function is called with multiple points, extended call graph will put this function for multiple nodes.In a traditional call graph, different call points point to the same node. In figure 2 as an example of traditional call graph comparison with the extended call graph, Open64 generates extended call graph during the interprocedural analysis by adding redundant nodes.

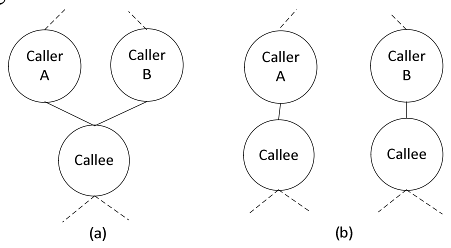


Fig.2 A extended call graph is an unfolded version of a traditional graph,(a) in a traditional function, the call graph nodes sharing the function call,(b) in an extended call graph, the function nodes having multiple function calls.

3.3 Call graph traversal

By checking the program each function body, Open64 add a side from the function call to the called function for each function call site. In such way to traverse the entire program source code, we get the call and the called relationships between all of the functions, generating the function call graph G = (N, E) of node and edge information, where N represents the function in the program, E represents the function call relationship [12].

Table 1 Call graph traversal sequence in Open64

structure	traversal sequence	specific order
PREORDER	preorder traversal	center, right ,left
POSTORDER	postorder traversal	right, left,center
LEVELORDER	level traversal	Top level to low level, each layer from right to left
DONTCARE	postorder traversal	left ,right, center

Table 1 describes the different traversal methods of relevant information.Because of the function call graph traversal has no effect on the validity of the context-sensitive detection algorithm, we choose the preorder traversal as the access of each node of function call graph.

4. Context sensitivity exploration algorithm

Source level context sensitivity detection algorithm is proposed in this paper, mainly traverses the acquired function call graph, to sort out the relationships between each call and called function nodes and gets the context sensitive information CONTEXT_SENSITIVE_INFO and the call information CALL_INFO. With statistical information, it is convenient to accurate software behavior model.structure description as shown in figure 3:

```
typeof struct{
int deep;
int total _ callsite _ num;
int insen _ callsite _ num;
float insen _ rate;
int insen _ callee _ num;
float ave _ insen _ callsite _ num;
int insen _ max_ deep;
}CONTEXT_SENSITIVE_INFO
```

```
typeof struct{
int callsite _ deep;
char callsite _ name[];
int callee _ deep;
char callee _ name[];
}CALL_INFO
```

Fig.3 The structures of CONTEXT_SENSITIVE_INFO and CALL_INFO

Figure 4 is the context sensitive detection algorithm pseudo code, this algorithm analyses sensitivity of the extended function call graph obtained by preorder traversal. L1-L5 is the initialization section. First, we initialize the node iterator, generate *context_sensitivity_info* to store sensitive information and *callee_info* to store relationship of the function call points. We set the maximum call point number *MAX_CALLSITE* to ensure maximum application space, prevent statistical information overflow. Algorithm process is described as follows:

(1) L6 is a loop structure, node iterator *cg_iter* traversal each node of the interprocedural call graph.

(2) L8, L9 is aimed at the current traverse node, using the function *Get_Sorted_Callsite_List* to obtain a list of the current nodes including the first level call points. The list saves the next level of information of the current node.

(3) L10 will traverse the current node and the next node level call edges and save to *IPA_EDGE_INDEX* structure. This call edge structure stores the directed call edges, distincts node call relations, at the same time, initializes directed edge iterator *ipa_edge_iter*.

(4) L12 through edge iterator *ipa_edge_iter* traverses the context information of the node and the call point, outputs the relationships of nodes which are called and the next level of nodes. Through the iterative operation, the algorithm completes the context sensitive analysis of all nodes.

(5) L13 to 15 on the iterator *ipa_edge_iter* iterative operation, update the context sensitive information *CONTEXT_SENSITIVE_INFO* and the *CALL_INFO*.

(6) L18, L19,after node iterator *cg_iter* and edge iterator *ipa_edge_iter* traversals are completed, the algorithm calculates the insensitive call point number information, insensitive probability, average context insensitive call points number and other information.

```
procedure   Context_Sensitivity_Analysis(File * fp, TRAVERSAL_ORDER order = PREORDER)
(1)    IPA_NODE_ITER cg_iter = Initial_IPA_Node_iter(IPA_CALL_GRAPH,order);
(2)    CONTEXT_SENSITIVITY_INFO context_sensitivity_info;
(3)    CALLEE_INFO callee_info[MAX_CALLSITES];
(4)    Initial_context_sensitivity_info(context_sensitivity_info);
(5)    int callee_func_num = Stat_Callee_func_Num(cg_iter);
(6)    for each cg_iter do
(7)      IPA_NODE * node = cg _ iter.Current();
(8)      vector<IPA_EDGE_INDEX> callsite_list;
(9)      callsite_list = Get_Sorted_Callsite_List(node, IPA_CALL_GRAPH);
(10)     vector<IPA_EDGE_INDEX> ipa_edge_iter = Initial_IPA_Edge_iter (callsite_list);
(12)     for each ipa_edge_iter do
(13)       context_sensitivity_info.totol_call_site_num++;
(14)       Accumulate_Callee_Info(callee_info);
(15)       context_sensitive_info.insen_max_deep=Calculate_Insen_Max_Deep(callee_info, IPA_CALL_GRAPH);
(16)     endfor
(17)   endfor
(18)   Calculate _ Insen _ Callee _ Info(context_sensitivity_info,callee_info);
(19)   Calculate _ Other _ Probability _ Info(context_sensitivity_info);
(20)   Fprint_Vobose(fp,context_sensitivity_info);
end;
```

Fig.4 Source code level context sensitivity exploration algorithm

5 Experiment and Analysis

The algorithm framework is based on Open64 5.0,adding the call relationship analysis for the entended function call graph.Experimental platform for the IBM 3850 server, frequency 2.0 GHz processor, 2 gb memory, L1 data cache to 32 KB, L2 cache is 256 KB, basic page for 8 KB. The operating system kernel for Linux 2.6.18, version for Redhat Enterprise AS 5.0.

5.1 correction validation

First, the experiment tests the ordinary procedures where the main program contains function call relations, mainly verify condition, loop, recursive three control structures in the program. Results are as follows:

(1) For the general order of execution of the program, the algorithm has a better ability to determine the function call context sensitive information.

(2) In the case of branch or loop statement contained in the main function or sub-function, as shown in Figure 5. Function call point sensitivity relationship outputs the call depth 3 of the *for* loop. Branch statements and sequence statements get the same context sensitive information.

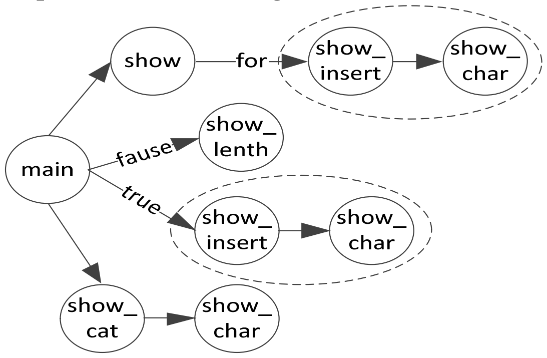


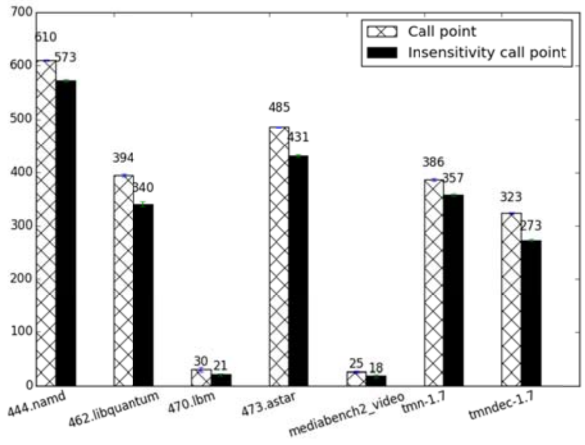
Fig.5 Insensitive function calls in the branch and loop structure

(3) Recursive function is a function call site calls itself in the implementation process. Whether it's called directly or using recursive functions in a conditional structure, the sensitivity of the depth is 2.

All of above,the algorithm can correctly acquire function call point context sensitive information.

5.2 Effectiveness Test

Effectiveness experiment uses the C program in Spec2006 and NPB3.3.1 as experimental samples. Test results as shown in figure 6 and 7:



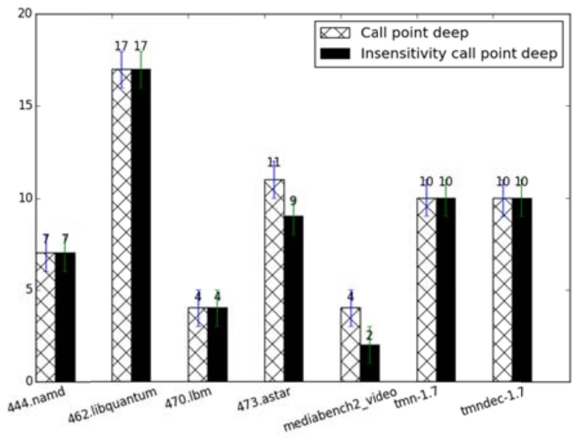


Fig.6 context insensitive call point statistics   Fig7 The maximum insensitive call point depth

From Fig.6,the context insensitive call points proportion is higher. In the process of establishment of behavior model, you need to consider context sensitive modeling methods such as PDA model, HFA model, Dyck model, abstract stack model combined with FSA model and other methods, will be able to more accurately describe the behavior of the software model.

From Fig.7,results in addition to *473.astar* and *mediabench2_video* in the maximum insensitive depth is less than the maximum depth of the function call graph, the others has a same maximum depth of function call point. The main reason is some points in a function call after the non-leaf node is called, and has been called in the leaf node, will have an impact to the maximum insensitive depth.

5.3 Comparison of procedure modeling methods

For the validity test subjects C language program in Spec2006 and NPB3.3.1, We are modeling the source code level of behavior. This paper adopts the method of combining FSA with PDA for the source code of behavior mode, comparing the accuracy of this model with the single using the FSA model, Dyck model or abstract stack to model. The results as shown in table 2:

Table 2 software modeling method comparison in Spec2006 and NPB3.3.1

experimental subject	FSA+PDA hybrid model	FSA model	Dyck model	abstract stack model
444.namd	98.3%	73.9%	89.3%	85.3%
462.libquantum	97.4%	75.5%	88.7%	83.7%
470.lbm	98.1%	70.7%	85.1%	81.6%
473.astar	97.3%	72.5%	86.9%	84.9%
mediabench2_video	89.7%	90.8%	88.5%	86.4%
tmn-1.7	93.6%	87.8%	87.9%	85.8%
tmndec-1.7	96.7%	80.9%	90.3%	85.7%

As seen from the table 2, the FSA and PDA mix-context sensitive model’s accuracy is higher. This shows that the context sensitive detection algorithm can guide the software behavior modeling a more accurate model.

6 The conclusion

In this paper, we propose a source code level context sensitivity exploration algorithm, can accurately analyze the function call points’ context sensitive information. Through the basis function calls tests, the sensitivity of detection of the larger source code function call points in Spec2006 for and NPB3.3.1, we can prove that the algorithm can obtain correct and effective function sensitive information. The comparison of modeling methods shows: the method of FSA and PDA hybrid model has a higher than accuracy other modeling methods. The next step of work:

(1) The algorithm utility platform for further expansion, in addition Open64 other compilers can also be integrated the sensitivity detection.

(2) We will further expand the application scope of the proposed algorithm, to use it to detect the context sensitivity of executable code, assembly code, etc.

(3) We can expand the function call relationship of the context sensitive information to more diverse software behavior modeling, to improve the accuracy and efficiency of software analysis.

Acknowledgements

This work was financially supported by the National Natural Science Foundation of China (No.61472447), the National High Technology Research and Development Plan(863)(2009AA012201).

References

[1] Fen Tao, Zhiyi Yin, Jianming Fu.Software behavior model based on system calls.[J].Computer Science，2010，04:151.

[2] Wagner D, Dean D. Intrusion detection via static analysis[C]//Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on. IEEE, 2001: 156-168.

[3] Xing Liu.The Homologous Analysis of Malware Based on function-call Graph[D].National University of Defense Technology，2012.

[4] Feng H H, Kolesnikov O M, Fogla P, et al. Anomaly detection using call stack information[C]//Security and Privacy, 2003. Proceedings. 2003 Symposium on. IEEE, 2003: 62.

[5] Zhen Li, Junfeng Tian, Xiaohui Yang.Program behavior monitoring based on system call attributes[J].Journal of Computer Research and Development,2012,08:1676-1684.

[6] Bond M D, McKinley K S. Probabilistic calling context[C]//ACM SIGPLAN Notices. ACM, 2007, 42(10): 97-112.

[7] Giffin J T, Jha S, Miller B P. Efficient Context-Sensitive Intrusion Detection[C]//NDSS. 2004.

[8] Wen Li, Yingxia Dai, Yifeng Lian,et al. Context sensitive host-based IDS using hybrid automaton[J].Journal of Software,2009,01:138-151.

[9] Tianwei Sheng, Wengguang Chen, Weimin Zheng. A context-sensitive pointer analysis phase in Open64 Compiler[C].//2nd Annual Workshop on Open64 in Conjunction with IEEE/ACM International Symposium on Code Generation and Optimization(CGO).2009.

[10]Reandy Allen,Ken Kennedy.Optimizing Compilers for Modern Architectures,A Dependence-Based Approach[M].USA:Elsevier Science,2001.

[11] L.F.C.C.Mallens.A framework for data-acceress strategies in GPGPU programs[D].Technische Universiteit Eindhoven,2013.

[12]Alfred V.Aho,Monica S.Lam,Ravi Sethi,et al. Compilers:Principles,Techniques and Tools (Second Edition)[M]. Bejing:China Machine Press,2009.