A GSoC 2013 Proposal

Inter-Procedural Program Slicing in LLVM

Mingliang Liu, *Tsinghua University*

This is a GSoC 2013 proposal for LLVM project. Program slicing has been used in many applications, the criteria of which is a pair of statement and variables. I would like to write an inter-procedural program slicing pass in LLVM, which is able to calculate C program slices of source code effectively. There is no previous work implemented in LLVM, which considers both the dynamic program slicing and source code of the sliced program.

Background

Program slice contains all statements in a program that directly or indirectly act the value of a variable occurrence [5]. Program slicing has been used in many applications, e.g., program verification, testing, maintenance, automatic parallelization of program execution, automatic integration of program versions. While it's straightforward to implement the slicer in the back-end of compiler using SSA form, the source code of the original program instead of intermediate representation is preferred in most cases. Moreover, we can further narrow the notion of *slice*, which contains statements that influence the value of a variable occurrence for special program inputs. This is referred as dynamic program slicing [1]. However, there is no previous work implemented in LLVM which solved the two problems.

Motivation

There are two public projects which implement the backwards static slicing in LLVM.

Giri Written by John Criswell from UIUC, a sub-project of LLVM. The Giri code contains the static backwards intra-procedure slicing passes, and runs with an older version of LLVM. It also only backtracks until it hits a load. Additional code must be written to backtrack further to find potentially reaching stores.

LLVMSlicer This implementation is a complete static backwards slicer from Masaryk University. It works on the well defined data and control flow equations in a white paper by F. Tip [4]. However, this code was written for specific purpose, thus it's not general enough to be use by others. They implemented the Andersen's alias algorithm [2], callgraph, and modifies analysis to support the slicer, instead of using the LLVM APIs.

They eliminate the useless IR statements and keep the ones affect the values of the criteria. However, neither of them generates the compilable source code slice, which is heavily needed in reality. There are several ways to do this. One is to generate the source code from sliced IR using `llc` tool. The issue is that the IR is not concerned with high-level semantic. The generated source code is different from the original program and not suitable for human reading. Another approach is deleting the source code according to sliced IR, with line number information (in metadata of each instruction). The naive script deleting sliced source code one by one fails to handle tricky cases. I think a better source code slicer is to take use of the AST info.

Plan

There are three main steps of this proposal.

First, borrow an implementation of static backwards slicing from Giri or LLVMSlicer, and use the LLVM callgraph, mod/ref and alias interfaces as much as possible.

Second, implement the dynamic program slicing using the approach 3 in the paper [1].

Third, generate the source code of the sliced program. To make the sliced source code compile directly, we need to employ clang front-end.

Table 1: *Plan of the project*

Work	Weeks
Investigate	1
Static inter-procedure slicing	3
Dynamic slicing	3
Source code generating	3
Slice large programs	2
Scrub code, write tests	1

Conclusion

I'd like to write an inter-procedural program slicing pass for LLVM, which considers both the dynamic program slicing and source code of the sliced program. The final result of this summer of code is to make this pass work effectively and documented well. Further, I'll write test cases and behave as the active maintainer for this project. My long-term plan is to add more features, e.g. Objective-C/C++ support, thing slicing [3], to this project.

Any comment is highly appreciated.

References

[1] H AGRAWAL. Dynamic Program Slicing. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 1990.

[2] L.O. Andersen. *Program analysis and specialization for the C programming language.* PhD thesis, University of Cophenhagen, Germany, 1994.

[3] M. Sridharan, S.J. Fink, and R. Bodik. Thin slicing. In *Proceedings of the ACM SIGPLAN conference on Programming Language Design and Implementation*, volume 10 of *PLDI'07*, pages 112–122, 2007.

[4] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121–189, 1995.

[5] M. Weiser. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering*, ICSE, pages 439–449. IEEE, 1981.