Incremental Decompilation of Loop-Free Binary Code: Erlang[1]

Gregory Morse, Dániel Lukács, Melinda Tóth
Department of Programming Languages and Compilers, Faculty of Informatics, Eötvös Loránd University
`gregory.morse@live.com dlukacs@caesar.elte.hu toth_m@inf.elte.hu`

In an era with a lot of advancement in areas such as incremental algorithms and boolean satisfiability (SAT) solvers, the question of how to properly structure a decompilation tool[1] to function in a completely incremental manner has remained an interesting problem.

This paper will present a concise algorithm and structuring design pattern for byte code which has a loop-free representation, as is seen in the Erlang language[2].

Various concepts and tools maintain the incremental cascading of effects. An analysis of semantic equivalence of byte code in a meta-data enhanced abstract syntax tree (AST) representation for any language allows for a cross-language approach. The importance of classifying side effects, required scenarios for variable emission and nearly inexpressible byte code operations is demonstrated. The incremental maintenance of dominator trees[3], reachability, and common ancestors are discussed as with minimal processing at merge nodes.

Data interfaces encapsulate the graph structure containing basic blocks, and another is used for the enhanced AST. Algorithms are considered for overall decompilation, dealing with edges not expressible in the target language, merge nodes and their optimized processing and a minimal variable emission.

2 scanning algorithms for overall decompilation are studied from traditional to a new one. The nuances highlight the technical challenge of achieving a consistent incremental algorithm.

Since code copying is a technique which has exponential growth consequences in complexity, simplifications for boolean short circuits are considered. The clean up of the AST is itself a crucial element of the decompiler for a readable and usable decompiled output.

References

[1] Cristina Cifuentes. *Structuring decompiled graphs*, pages 91–105. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[2] Dániel Lukács and Melinda Tóth. Structuring erlang beam control flow. In *Proceedings of the 16th ACM SIGPLAN International Workshop on Erlang*, Erlang 2017, pages 31–42, New York, NY, USA, 2017. ACM.

[3] Vugranam C. Sreedhar and Guang R. Gao. A linear time algorithm for placing ϕ-nodes. In *Proceedings of the 22Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, pages 62–73, New York, NY, USA, 1995. ACM.

[1]The project has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).