Purple Dragon book - Chapter 12.5 & 12.6 (Context-Insensitive Interprocedural Analysis & Context-Sensitive Pointer Analysis)

Ondřej Hamada (xhamad00), Marek Hložánka (xhloza01)

Our presentation discusses interprocedural pointer analysis. By interpocedural analysis we mean data-flow analysis that tracks information across procedure boundaries. Because of this we have to deal with method invocations.

In the first part we discuss context-insensitive analysis. Such analysis does not take history of method calls into consideration. Also the parameters and returned values are modeled by copy statements. Still, we have to find a way to determine the type of the receiver object. This is getting complicated in real-world programs that use object hierarchies and include large libraries. The analysis tends to become slow and imprecise under such conditions.

In order to do the analysis we need to compute the call targets. Call targets are represented by call graphs that are bipartite graphs with nodes for call sites and procedures and edges going from call site to procedure if that procedure can be called from that site. Call sites are points in a program from which procedure is being called.

Problem is that this task requires the knowledge of what the variables point to, but this data can be computed only when the call graphs are known. Solution lays in discovering the call graphs on the fly while computing the points-to set. This analysis is running until no new call targets or points-to relations are found.

For computation of the call graph we introduce several 'Datalog' rules. 'Datalog' is language that presents simple notation of if-then rules that can be used to describe data-flow analysis at a high level. The notation of rules in 'Datalog' are similar to Prolog programming language. The newly proposed rules for computation of the call graph will be further described in our presentation.

Second part of our presentation focuses on context-sensitive pointer analysis. Taking context into consideration brings additional difficulties. The summaries of points-to information are getting too large – usually up to the size that makes it impossible to do any computation over it. To overcome this problem we use cloning-based context-sensitive analysis.

Cloning-based context-sensitive analysis means that once we establish the different contexts in which a procedure can be called, we can imagine that there is a clone of each procedure for each context. So in fact we use context-insensitive analysis to work as a context-sensitive. Product of this analysis is cloned call graph.

In a cloned call graph we must deal with recursive functions because they cause the number of possible call strings to be infinite. To avoid that, we have to find the mutually recursive functions. Mutually recursive functions are formed by strongly connected components (SCC) in call graph, so we have to rule out the context of any calls within the SCC to other functions in the same SCC. Complete algorithm and improved 'Datalog' rules with examples will be shown in our presentation.

The proposed solution still lacks the ability to handle large programs and ignores object sensitivity. Also the count of computed contexts is still large so we have to use binary decision diagrams in order to represent them, but all of this problems are beyond the scope of our presentation.