Abstract

This presentation discusses interprocedural analysis, which is important for solving optimization problems. Pieces of language codes Java and C will be used as examples.

It works with information from the caller, which call a procedure, to its callees (called procedures) and vice versa.

In this presentation are explained individual terms such as call graphs and call strings. Call graphs means set of nodes and edges for a program. One node for each procedure in the program. One node for each call site, which is a place in the program where a procedure is invoked. Edges join every call site with procedures that calls. There are two call ways. Direct – for C and Fortran, where edges are created statically. And indirect – for object-oriented language, where edges are created dynamically by using virtual method. Call strings call string is a string of call sites on the stack. Calling context is defined by the contents of the entire call stack. Call site is a place, from where is called a procedure.

Then interprocedural analysis is explained on examples in part context sensitivity, associated with parts cloning-based context-sensitive analysis which clones a procedure and summary-based context-sensitive analysis which progressively replace a procedure with constants.

Context sensitivity says that behavior of each procedure depends on context in which it is called. Then context-insensitive analysis is inaccurate analysis, which take each call and return statement as ”goto” operations. Assignment statements are added to assign each actual parameter to its corresponding formal parameter and to assign the returned value to the variable receiving the result.

Principe of cloning-based context-sensitive analysis is, that we clone procedure conceptually, for each unique context of interest one. Then we can then apply a context-insensitive analysis to the cloned call graph. In reality, we do not need to clone the code, we can simply use an efficient internal representation to keep track of the analysis results of each clone. Summary-based context- sensitive analysis is an extension of region-based analysis. Each procedure is represented by a concise description (”summary”) that encapsulates some observable behavior of the procedure.

The primary purpose of the summary is to avoid reanalyzing a procedure’s body at every call site that may invoke the procedure. The only difference from the intraprocedural version is that, in the interprocedural case, a procedure region can be nested inside several different outer regions. The analysis consists of two parts. A bottom-up phase, which computes a transfer function to summarize the effect of a procedure, and a top-down phase, which propagates caller information to compute results of the callees. Instead of cloning a function, we could also inline the code. Inlining has the additional effect of eliminating the procedure-call overhead as well. Principe of summary-based context-sensitive analysis is that in the presence of recursion, we first find the strongly connected components in the call graph. In the bottom-up phase, we do not visit a strongly connected component unless all its successors have been visited. For a nontrivial strongly connected component, we iteratively compute the transfer functions for each procedure in the component until convergence is reached (no more changes occurs).

At the end of the presentation a Virtual method invocation is presented, which principle is inline sections of code, which are called most often. For optimization object-oriented programs with many small methods, we can use a method invocation, where we do not know, how many method named m refers in an invocation such x.m().(Java). In common optimization is to profile the execution and determine which the common receiver types are. Then we can inline the methods that are most frequently invoked. The code must include a dynamic check on the type and can execute the inlined methods. If we have all source code at compile time, we can perform an interprocedural analysis to determine the object types. If the type for a variable x turns out to be unique, then a use of x.m() can be resolved.

Last things shown here are interesting computers vulnerabilities - SQL injection and buffer overflow and how to defend against them. SQL Injection is used in database application. It is one of the most popular forms vulnerability, because most values, entered by user, are substituted to SQL query. If hacker knows about structure of database, he can write part SQL string instead of required string. Then he can do whatever he wants with the database. Prevention is to check all strings from user. Buffer overflow occurs when data are written beyond the intended buffed and

manipulates the program execution. This causes programs, which do not implicitly check boundary of arrays (C and C++).