Abstract

This work presents a new approach to detect may-aliases within ANSI-C programs. A may-alias occurs if two variables (or more complex expressions) use the same memory location. Aliases may be created by call-by-reference function calls or by pointer usage. In the context of C programs alias analysis becomes very complex because of the extensive pointer usage and the very few restrictions concerning pointer manipulations.

Within this work it is described how the effects of a program can be summarised by so-called function interface graphs which are a static representation of the memory locations and the values stored at these locations. Based on standard techniques (function call and return statement normalisation, control flow graphs, static single assignment form), the intraprocedural step creates a function interface graph for each function individually by computing and merging the information for all its basic blocks according to the CFG structure. Afterwards these graphs will be reduced to their externally visible effects. This reduces the size of these graphs and will hence allow the following computations to be carried out more efficiently. Within the next step the reduced graphs will be merged according to the corresponding functions and their calls (interprocedural analysis). This is done by ignoring indirect function calls (through function pointers) first, and processing these calls only if a function that could be called by one of these calls is detected. In case functions are called by more than a single function call, the calling context will be taken into account as far as possible.

A major benefit of our algorithm is that it can be applied to real ANSI-C programs since it makes only a few restrictions, i.e. effects of assembler code, interrupts, volatile attributes and I/O-based aliases. It deals with structures and unions, arbitrary pointer usage, type casts and function pointers. Under these circumstances, the algorithm cannot benefit from type information and, hence, the function interface graphs are based on a low-level memory representation. The algorithm was implemented using the SUIF compiler, and it has been successfully applied to a set of non-toy C programs.