

Interrupt Handlers in Dead Variable Analysis

Micah Lewis, msl@byu.edu
Department of Computer Science
Brigham Young University
Provo, Utah, USA

Technical Report SMC-BYU-0105

October 25, 2005

Abstract

Static dead variable analysis has long been used to reduce the complexity of model checking software. Identifying dead variables in interrupt-driven code using context-insensitive static analysis is difficult because the interrupt handler has many call points and many return points. This fractures the set of basic blocks and adds many infeasible paths to the control flow graph. The problems caused by interrupts in context-insensitive dead variable analysis can be reduced by removing the interrupt handler from the control flow graph and using a summary of which variables are used and defined in the interrupt handler instead. The primary advantage of this approach is that it avoids fracturing the control flow graph.

Allowing interrupts in a program changes the control flow graph (CFG) constructed for a program by introducing execution paths. Figure 2 shows the CFG that would result if an interrupt handler were added to the CFG in Figure 1. When an interrupt occurs, the handler is called immediately following the current instruction. We assume interrupts can occur at any time, the CFG must include an edge from every basic block to the handler and from the handler to every basic block. It also forces basic blocks to be exactly one instruction long.

The CFG as shown in Figure 2 is an over-approximation of the true control flow because it includes interrupts in its construction. In reality, the additional paths of execution do not exist. For example, if the interrupt occurred after executing the instruction in $B1$, the interrupt handler at Bi would execute and then return control to the next instruction at $B2$. But from the CFG it appears that the program could return to any instruction.

A dead variable analysis on a CFG that over-approximates the actual control flow results in dead variable sets that under-approximate the actual dead variables in each basic block. A more accurate CFG is shown in Figure 3 in which the interrupt handler is duplicated to show when it might occur in reality and what the true flow of control is. The modified CFG more accurately depicts the flow of control and removes many extraneous paths.

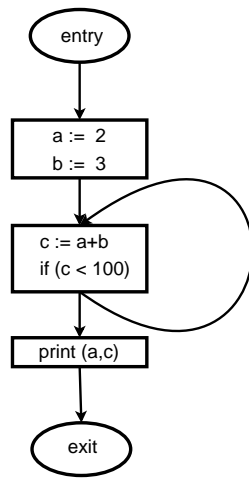


Figure 1: A CFG without interrupts

The CFG in Figure 3 is not, however, ideal since the size of each basic block is no greater than a single instruction long. This unnecessarily increases the number of basic blocks that must be considered in the analysis. With more basic blocks, the amount of computation per analysis increases. At the same time, the influence of the interrupt handler causes any global variable shared by the interrupt handler to be considered live until the end of the program.

A new dead variable analysis (DVA) might avoid the extra computation while deriving the same information including interrupt handlers in the CFG by considering the set of all used variables from interrupt handlers as used variables in every basic block. By doing so, the dead variable calculation always makes the variables live and only needs to consider the basic blocks shown in Figure 1.

This new DVA uses a summary of the interrupt handler to include the influence of the interrupt handler on the DVA without including the interrupt handler. This idea is similar to use of procedure (or function) summaries in inter-procedural data flow analysis.

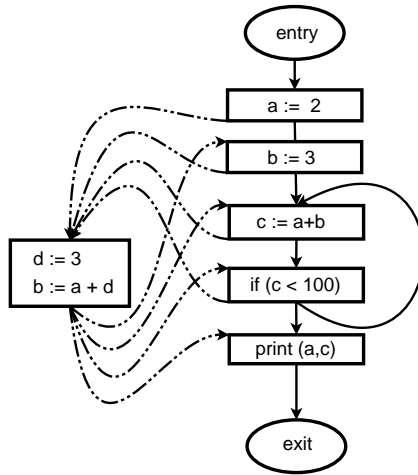


Figure 2: A CFG with interrupts

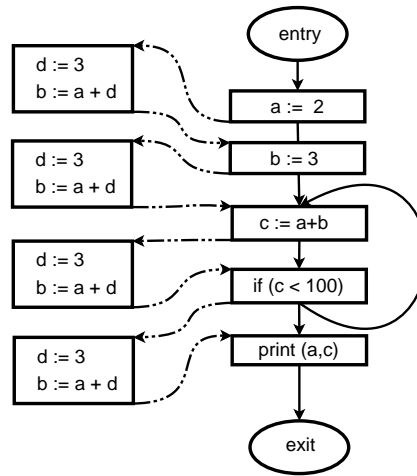


Figure 3: A more accurate CFG of a program with interrupts