Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 101

A Simple and Scalable Static Analysis for Bound Analysis and Amortized Complexity Analysis

Moritz Sinn, Florian Zuleger and Helmut Veith (TU Wien) [1]

Automatic methods for computing bounds on the resource consumption of programs are an active area of research. In [SZV14] we present the first scalable *bound analysis* for imperative programs that achieves *amortized complexity analysis*. Our techniques can be applied for deriving upper bounds on how often loops can be iterated as well as on how often a single or several control locations can be visited in terms of the program input.

The majority of earlier work on bound analysis has focused on mathematically intriguing frameworks for bound analysis. These analyses commonly employ general purpose reasoners such as abstract interpreters, software model checkers or computer algebra tools and therefore rely on elaborate heuristics to work in practice. Our work [SZV14] takes an orthogonal approach that complements previous research. We propose a bound analysis based on a simple abstract program model, namely *lossy vector addition systems with states*. We present a static analysis with four well-defined analysis phases that are executed one after each other: program abstraction, control-flow abstraction, generation of a lexicographic ranking function and bound computation.

```
    void main(uint n) {
      int a = n, b = 0;
l1:   while (a > 0) {
         a--; b++;
l2:      while (b > 0) {
            b--;
l3:         for (int i = n-1; i > 0; i--)
               if (a > 0 && ?) {
l4:               a--; b++;
    }  }  }  }
```

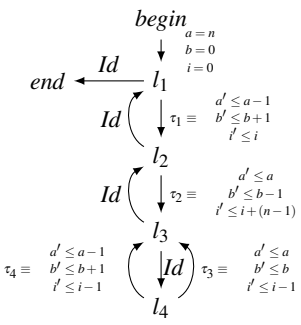

Figure 1: Our running example, '?' denotes non-determinism (arising from a condition not modeled in the analysis). On the right we state the lossy VASS obtained by abstraction, *Id* denotes $a' \leq a, b' \leq b, i' \leq i$.

The example presented in Figure 1 (encountered during our experiments) is challenging for an automated bound analysis: (C1) There are loops whose loop counter is modified by an inner loop: the innermost loop modifies the counter variables a and b of the two outer loops. Thus, the inner loop *cannot be ignored* (i.e., cannot be sliced away) during the analysis of the two outer loops. (C2) The middle loop with loop counter b requires a *path-sensitive* analysis to establish the linear loop bound n: it is not enough to consider how

[1] Supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grants PROSEED and ICT12-059.

often the innermost loop can be executed (at most n^2 times) but rather how often the if-branch of the innermost loop (on which b is actually incremented) can be executed (at most n times). (C3) Current bound analysis techniques cannot model *increments* and instead approximate increments by *resets*, e.g., approximate the increment of b by an assignment to a value between 0 and n (using the fact that n is an upper bound of b)! Because of this overapproximation existing bound analysis techniques fail to compute the linear loop bound n for the middle loop. We illustrate the main steps of our analysis:

1. Program Abstraction: First, our analysis abstracts the program to the VASS depicted in Figure 1. We are using *parameterized* VASSs, where we allow increments that are symbolic but constant throughout the program (such as $n-1$). We extract lossy VASSs from C programs using simple invariant generation and symbolic execution techniques (see [SZV14]).

2. Control Flow Abstraction: In [SZV14] we propose a new abstraction for bound analysis, which we call *control flow abstraction* (CA). CA abstracts the VASS from Figure 1 into a transition system with four transitions: $\rho_1 \equiv a' \leq a-1 \wedge b' \leq b+1 \wedge i' \leq i$, $\rho_2 \equiv a' \leq a \wedge b' \leq b-1 \wedge i' \leq i+(n-1)$, $\rho_3 \equiv a' \leq a \wedge b' \leq b \wedge i' \leq i-1$, $\rho_4 \equiv a' \leq a-1 \wedge b' \leq b+1 \wedge i' \leq i-1$.
CA effectively merges loops at different control locations into a single loop creating one transition for every cyclic path of every loop (without unwinding inner loops). This significantly simplifies the design of the later analysis phases.

3. Ranking Function Generation: Our ranking function generation (discussed in [SZV14]) finds an *order* on the transitions resulting from CA such that there is a variable for every transition, which decreases on that transition and does not increase on the transitions that are lower in the order. This results in the lexicographic ranking function $l = \langle a, a, b, i\rangle$ for the transitions $\rho_1, \rho_4, \rho_2, \rho_3$ in that order. Our soundness theorem (see [SZV14]) guarantees that l proves the termination of Figure 1.

4. Bound Analysis: Our bound analysis (discussed in [SZV14]) computes a bound for every transition ρ by adding for every other transition τ how often τ increases the variable of ρ and by how much. In this way, our bound analysis computes the bound n for ρ_2, because ρ_2 can be incremented by ρ_1 and ρ_4, but this can only happen n times, due to the initial value n of a. Further, our bound analysis computes the bound $n*(n-1)$ for ρ_3 from the fact that only ρ_2 can increase the counter i by $n-1$ and that ρ_2 has the already computed transition-bound n. Our soundness result (see [SZV14]) guarantees that the bound n obtained for ρ_2 is indeed a bound on how often the middle loop of Figure 1 can be executed.

Our bound analysis solves the challenges (C1)-(C3): CA allows us to analyze all loops at once (C1) creating one transition for every loop path (C2). The abstract model of lossy VASS is precise enough to model counter increments, which is a key requirement for achieving amortized complexity analysis (C3).

References

[SZV14] Sinn, Moritz; Zuleger, Florian; Veith, Helmut: A simple and scalable static analysis for bound analysis and amortized complexity analysis. In: CAV. Springer, pp. 745–761, 2014.