Loopus - A Tool for Computing Loop Bounds for C Programs

Moritz Sinn
TU Darmstadt
sinnmoritz@googlemail.com
and Florian Zuleger
TU Wien
zuleger@forsyte.tuwien.ac.at

Abstract

We describe the current state of our tool Loopus which computes loop bounds for C programs.

In this one-page abstract we describe the current state of our tool Loopus which automatically computes loop bounds for C programs. Loopus can also be seen as contribution towards solving the termination problem for C programs as it provides a conceptually simpler alternative to recently proposed methods [1]. The underlying reasoning of Loopus builds on earlier work of the second author [2] of which we give a short description below. In contrast to [2] which is targeted at .NET programs, our tool analyzes programs written in C, which pose a different challenge to static analyses.

Program abstraction. Following [2], we abstract loops to transition systems. For a given loop we enumerate all paths which start from its header and go back to its header. For every such path we construct a logical formula which overapproximates the possible state transitions along that path. The abstraction is done such that enough information about the program is kept to analyze its termination behavior. As the number of paths from the header back to the header can be infinite due to the presence of inner loops we replace inner loops beforehand by the transitive closure of their transition systems.

Transitive closure computation. In order to summarize inner loops we need disjunctive relational invariants which overapproximate their transitive closure. Instead of relying on the complexity-like assumption of [2], which enables abstract interpretation techniques to compute disjunctive invariants, we follow the size-change principle [4]. Inner loops are abstracted by sets of transitions which record only the monotonic behavior of the variables of the outer loop, i.e., if they de- or increase strictly or non-strictly. This ensures the finiteness of the transitive closure of inner loops. Therefore our work can be seen as a new way of applying the size-change abstraction to imperative programs.

Bound Computation. For every transition we identify expressions which constitute a bound on the number of times the transition can be repeated if no other transition is taken in between. An overall bound for the loop is constructed according to a set of rules on how to combine these local expressions.

Non-relational invariants. In order to compute bounds we need several non-relational invariants: shapes of data structures in the heap, alias sets of pointers, and upper and lower bounds on numerical expressions.

Implementation. Our tool uses the LLVM compiler framework [3]. We make use of slicing to remove the parts of the program irrelevant to the termination of the analyzed loop. We use a SMT solver to discharge queries about the program.

Benchmarks. We currently investigate benchmarks from the WCET community and the SPEC CPU2006 benchmark.

References

[1] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Termination Proofs for Systems Code. In *PLDI*, pages 415–426, 2006.

[2] Sumit Gulwani and Florian Zuleger. The Reachability-Bound Problem. In *PLDI*, 2010. to appear.

[3] Chris Lattner and Vikram S. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *CGO*, pages 75–88, 2004.

[4] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The Size-change Principle for Program Termination. In *POPL*, pages 81–92, 2001.