FuncTion: An Abstract Domain Functor for Termination*

(Competition Contribution)

Caterina Urban

ÉNS and CNRS and INRIA, France
urban@di.ens.fr

Abstract. FuncTion is a research prototype static analyzer designed for proving (conditional) termination of C programs. The tool automatically infers piecewise-defined ranking functions (and sufficient preconditions for termination) by means of abstract interpretation. It combines a variety of abstract domains in order to balance the precision and cost of the analysis.

1 Verification Approach

FuncTion is a prototype implementation of our analysis method and abstract domains described in [6,7,8].

Our analysis method follows the traditional approach for proving program termination by means of a well-founded argument or ranking function (i.e., a function from the states of a program to a well-ordered set whose value decreases during program execution). We build a ranking function for a program in an incremental way: we start from the program final states, where the function has value zero (and is undefined elsewhere); then, we add states to the domain of the function, retracing the program backwards and counting the maximum number of performed program steps as value of the function. In [2], Cousot and Cousot formalize this intuition into a sound and complete termination semantics, which is systematically derived by abstract interpretation of the program operational trace semantics.

In order to achieve an effective static analysis, we further abstract this semantics by means of piecewise-defined ranking functions. The analysis consists of two phases: a forward reachability analysis, followed by a backward termination analysis. Each phase proceeds by structural induction on the program syntax, iterating loops until stabilization. In case of nested loops, the analyses stabilize the inner loop for each iteration of the outer loop.

The forward analysis computes, at each program control point, an over-approximation of the set of program states that can be reached at these program points by considering all possible program executions. This provides a first over-approximation of the domain of the program ranking functions.

* The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement no. 269335 (ARTEMIS project MBAT) (see Article II.9. of the JU Grant Agreement).

C. Baier and C. Tinelli (Eds.): TACAS 2015, LNCS 9035, pp. 464–466, 2015.
DOI: 10.1007/978-3-662-46681-0_46

The backward analysis computes, at each program control point, a piecewise-defined ranking function whose domain is (a subset of) the set of reachable states identified by the forward analysis, and whose value represents an upper bound on the number of program execution steps remaining before termination. The starting point is the constant function equal to zero at the program final control point. The piecewise-definition of the ranking functions is semantic-based and dynamic: during the analysis, pieces are split by tests, modified by assignments, and joined when merging control flows. In order to minimize the cost of the analysis, a widening limits the number of maintained pieces. The domain of the ranking function at the program initial control point provides a sufficient precondition for program termination: all program executions starting from a state in the domain of the ranking function are terminating.

2 Software Architecture

FuncTion is written in OCaml. For parsing C programs, we use our own ad-hoc parser generated using Menhir[1]. The available abstract domains for the forward reachability analysis are the numerical abstract domains of intervals [1], octagons [5], and convex polyhedra [3] provided by the APRON library [4]. The abstract domains used for the backward termination analysis are implemented on top of the APRON library: the piecewise-defined ranking functions are represented as decision trees [8]; the nodes of the decision trees are interval, octagonal, or polyhedral linear constraints, and the paths towards the leaves induce the piecewise-definition of the ranking functions; the leaves of the decision trees represent the value of the ranking functions as affine functions or ordinal-valued functions [7]. For the competition, we have chosen convex polyhedra for the reachability analysis and polyhedral linear constraints for the decision trees in the termination analysis.

3 Strengths and Weaknesses

A strength of FuncTion is its modular architecture: a variety of abstract domains are combined in order to balance the precision and cost of the analysis. An immediate consequence is the potential for improvements of the analysis by simply adding new abstract domains to the analyzer. However, FuncTion is still a research prototype, and so far it lacks any abstract domain for shape analysis: it provides only a limited support for arrays and pointers. Therefore, FuncTion is able to analyze only 83% of the SV-COMP 2015 benchmark test cases.

Moreover, the analyzer fails to prove termination of a significant number of terminating tests cases mainly due to a naïve widening operator [6,8]. We have yet to integrate state-of-the-art widening operators.

We emphasize the soundness of the analysis, which is confirmed by the absence of reported false negatives (i.e., reported termination for a non-terminating program) on the benchmark of SV-COMP 2015. On the other hand, FuncTion does

[1] `http://cristal.inria.fr/~fpottier/menhir/`

not report non-termination (i.e., it does not answer FALSE) for now, which causes a fair loss of score.

Finally, we argue that the ability of FuncTion to find significative preconditions for program termination is an important feature, which unfortunately is not taken into account in the competition.

4 Tool Setup and Configuration

The competition candidate for SV-COMP 2015 can be downloaded from:

`http://www.di.ens.fr/~urban/sv-comp2015.zip`.

FuncTion is only participating in the `Termination` category of SV-COMP 2015.

The competition candidate can be invoked using the following call pattern:

```
./function <file>
```

where `<file>` is the path to the C file to be analyzed for termination of the function `main()`. The analyzer prints TRUE on the standard output in case it can successfully prove termination. Otherwise, it prints UNKNOWN.

5 Software Project and Contributors

FuncTion has been developed as part of the author's PhD thesis. A web interface is available: `http://www.di.ens.fr/~urban/FuncTion.html`.

Grateful acknowledgements go to Antoine Miné for publishing the source code of his prototype[2], which helped to speed up the initial development of FuncTion.

References

1. Cousot, P., Cousot, R.: Static Determination of Dynamic Properties of Programs. In: International Symposium on Programming, pp. 106–130 (1976)
2. Cousot, P., Cousot, R.: An Abstract Interpretation Framework for Termination. In: POPL, pp. 245–258 (2012)
3. Cousot, P., Halbwachs, N.: Automatic Discovery of Linear Restraints Among Variables of a Program. In: POPL, pp. 84–96 (1978)
4. Jeannet, B., Miné, A.: APRON: A library of numerical abstract domains for static analysis. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 661–667. Springer, Heidelberg (2009)
5. Miné, A.: The Octagon Abstract Domain. Higher-Order and Symbolic Computation 19(1), 31–100 (2006)
6. Urban, C.: The abstract domain of segmented ranking functions. In: Logozzo, F., Fähndrich, M. (eds.) Static Analysis. LNCS, vol. 7935, pp. 43–62. Springer, Heidelberg (2013)
7. Urban, C., Miné, A.: An abstract domain to infer ordinal-valued ranking functions. In: Shao, Z. (ed.) ESOP 2014 (ETAPS). LNCS, vol. 8410, pp. 412–431. Springer, Heidelberg (2014)
8. Urban, C., Miné, A.: A decision tree abstract domain for proving conditional termination. In: Müller-Olm, M., Seidl, H. (eds.) Static Analysis. LNCS, vol. 8723, pp. 302–318. Springer, Heidelberg (2014)

[2] `http://www.di.ens.fr/~mine/banal/`