ReFuncTion: Conditional Termination by Abstract Interpretation of Numerical C Programs
(Competition Contribution)

Naïm Moussaoui Remil and Caterina Urban

Inria & ENS | PSL, Paris, France
`{naim.moussaoui-remil,caterina.urban}@inria.fr`

Abstract. ReFuncTion is a static analyzer designed for proving conditional termination (resilience), and functional program properties expressed in Computation Tree Logic (CTL). The tool automatically infers piecewise-defined ranking functions and sufficient preconditions by means of abstract interpretation.

1 Verification Approach

ReFuncTion is an abstract-interpretation–based static analyzer for the verification of numerical C programs. It is a fork of the analyzer FuncTion [10,11], from which it inherits conditional termination analysis [11] and the verification of conditional Computation Tree Logic (CTL) properties [14]. ReFuncTion extends FuncTion with a new parser, support for signed machine integer, and non-recursive interprocedural analysis. Besides these programming-features extensions, ReFuncTion provides two new analysis: one for conditional termination resilience [7] and one for computing minimal set of variables that need to be controlled (i.e., constrained) to satisfy a given CTL property of interest [8].

1.1 Backward (Termination) Analysis

ReFuncTion proves termination (and other properties) by inferring a piecewise-defined ranking function [4,9], i.e., a function mapping program states to a well-ordered set and whose value strictly decreases during program execution.

The analysis implements the abstract-interpretation-based framework of Urban et al. [11,12], which soundly approximates the best ranking-function semantics of Cousot and Cousot [3]. It is performed backwards. For termination, ReFuncTion starts from the final program control point with a ranking function defined as the constant function `zero`. At each control point, it computes a piecewise ranking function whose domain under-approximates the set of states from which termination is guaranteed, and whose values over-approximate the remaining number of execution steps. The domain obtained at the initial control point therefore provides a sufficient condition for program termination.

1.2 Decision Tree Abstract Domain

The decision tree abstract domain (denoted $\mathcal{T}$) is used in ReFuncTion to represent and compute piecewise-defined ranking functions.

An element $t \in \mathcal{T}$ is a piecewise-defined partial function represented by a full binary tree, where each node — denoted $\textsc{node}\{c\}\colon t_1, t_2$, with $t_1, t_2 \in \mathcal{T}$ — is labeled by a constraint $c \in \mathcal{C}$ over the program variables (where $\mathcal{C}$ is the set of allowed constraints, e.g., linear constraints), and each leaf — denoted $\textsc{leaf}\colon f$ — is labeled by a function $f \in \mathcal{F}$ of the program variables (where $\mathcal{F}$ is the set of allowed functions, e.g., affine functions). Nodes recursively partition the space of possible values of the program variables: constraint labeling nodes are satisfied by the left subtrees of the nodes, while the right subtrees satisfy their negation. The leaves of the tree represent the value of the function within each partition. The partitioning is dynamic: during the analysis, partitions (resp. decision nodes and constraints) are split (resp. added) by tests, modified by variable assignments and joined (resp. removed) when merging control flows. In order to minimize the cost of the analysis, a widening limits the height of the decision trees and the number of maintained partitions. Figure 1 shows an example of a decision tree. The leaf with value $\bot_F$ explicitly represents the undefined partition of the (partial) function. Undefined functions can also be represented by the leaf value $\top_F$ in case of an irrecoverable loss of precision of the analysis [2]. The decision tree in Figure 1 represents a function with value 1 when $z < x$ (at most one step to termination), with value 3 when $z \geq x \land z + c < x$ (at most three steps to termination) and undefined otherwise (termination is not proved when $z + c \geq x$).

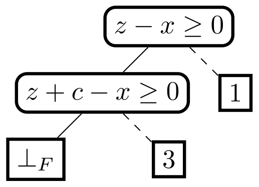

Fig. 1: Example of decision tree inferred by ReFuncTion.

1.3 Forward Reachability Analysis

ReFuncTion optionally performs a forward reachability analysis that computes an over-approximation of the reachable states at each program control point. When executed beforehand, this forward analysis restricts the state space by discarding unreachable states, thereby improving the precision analysis results.

1.4 New Features

ReFuncTion extends FuncTion to analyze programs manipulating signed integers. This extension relies on the assumption that signed integer overflows do not occur. Consequently, signed machine integers are modeled as mathematical integers constrained to their respective type bounds.

While previous versions of ReFuncTion relied on function inlining, we have now implemented a context-insensitive interprocedural analysis. Operating backwards, the analysis infers a decision tree t_1. Upon encountering a function call, ReFuncTion analyzes the callee independently to compute an associated decision tree t_2. The decision trees t_1 and t_2 are then joined by keeping paths defining the same sufficient preconditions and summing their associated values. This idea was originally proposed in [11] but was not implemented. In future versions, we plan to leverage function summaries to achieve a compositional analysis.

2 Software Architecture

ReFuncTion consists of 20 000 lines of OCaml code, and uses the parser of Mopsa [6] (available through its opam packages[1]) to parse C programs. Only numerical programs, without bitwise operations, goto/break, unsigned machine integers and recursive functions are supported by our analyses. ReFuncTion returns a special error for any program using an unsupported feature, `true` when termination holds for any program state at the initial program control point, and `unknown` when for some initial program state the analysis is inconclusive. The analyzer does not generate witnesses.

The available abstract domains for the forward reachability analysis are those provided by the Apron library [5]. The decision tree abstract domain used for the backward termination analysis is implemented on top of the Apron library; the internal nodes of these trees correspond to univariate or multivariate linear constraints, and the leaves of the decision trees represent the value of the ranking functions as affine functions or ordinal-valued functions [13]. For the competition, ReFuncTion used convex polyhedra both for the reachability analysis and for the decision trees in the termination analysis.

3 Strengths and Weaknesses

Strengths. ReFuncTion is a *sound static analyzer*, therefore for the competition it produced no incorrect results [1]. We report 206 timeouts, 200 of which are due to the tasks in the 'eca-rers2012' benchmarks, which implement the same programming pattern. As the timeouts are caused by only one kind of program, it showcases the ability of ReFuncTion to converge and eventually produce results within the time limit. Besides, ReFuncTion produces results within reasonable time, with an average of CPU time of $2.7\,s$ for the 111 correct results and $54s$ for the 131 files for which it is inconclusive and returns `unknown`. Finally, we emphasize that the ability of ReFuncTion to find meaningful preconditions for program termination is an important feature, which unfortunately is not taken into account in the competition.

[1] `https://opam.ocaml.org/packages/mopsa/`

Weaknesses. Currently, ReFuncTion is unable to analyze most of the programs. It parses and reports an unsupported-feature error for 1926 tasks out of the 2384 tasks in the Termination category. ReFuncTion mainly suffers from the lack of sound abstract domains for non-numerical program features, e.g., heap-manipulating programs. The termination analysis returns `unknown` for 101 tasks for which termination holds. This is mostly due to a naïve widening operator [11] in the current implementation of the decision tree abstract domain. This could be addressed through state-of-the-art widening [2] or by developing new widening operators. Similarly, the absence of non-linear numerical abstract domains limits the precision of ReFuncTion.

4 Tool Setup, Configuration and Contributors

The analyzer code source and a Docker image are also available at our GitHub repository[2].

A `README.md` file describes the instructions to build the executable `main.exe` using the `dune` build system. A sample run command is:

```
./main.exe filename.c -domain polyhedra
```

ReFuncTion has only participated in the Termination category of SV-COMP 2026.

ReFuncTion is developed at École Normale Supérieure and Inria Paris by Naïm Moussaoui Remil under the supervision of Caterina Urban. We thank all the contributors of FuncTion [10,11] from which we forked to develop ReFuncTion.

Data Availability. ReFuncTion's competition version is distributed as a Zenodo archive[3]. The tool-info module and the benchmark definition files for SV-COMP 2026 are named `function-res`.

Acknowledgements. This work has benefitted from the participation of the authors in the Dagstuhl Seminar 25421 "Sound Static Program Analysis in Modern Software Engineering".

References

1. D. Beyer and J. Strejček. Evaluating software verifiers for C, Java, and SV-LIB (report on SV-COMP 2026). In *Proc. TACAS (2)*, LNCS 16506. Springer, 2026.
2. Nathanaël Courant and Caterina Urban. Precise Widening Operators for Proving Termination by Abstract Interpretation. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 136–152, 2017.

[2] `https://github.com/naim-mr/refunction`
[3] https://zenodo.org/records/17525389

3. Patrick Cousot and Radhia Cousot. An Abstract Interpretation Framework for Termination. In John Field and Michael Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 245–258. ACM, 2012.
4. Robert W. Floyd. Assigning Meanings to Programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.
5. Bertrand Jeannet and Antoine Miné. Apron: A library of numerical abstract domains for static analysis. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*, pages 661–667. Springer, 2009.
6. Raphaël Monat, Abdelraouf Ouadjaout, and Antoine Miné. Mopsa-c with trace partitioning and autosuggestions (competition contribution). In Arie Gurfinkel and Marijn Heule, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, TACAS 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3-8, 2025, Proceedings, Part III*, volume 15698 of *Lecture Notes in Computer Science*, pages 229–235. Springer, 2025.
7. Naïm Moussaoui Remil and Caterina Urban. Termination Resilience Static Analysis. In *VMCAI 2026 - 27th International Conference on Verification, Model Checking, and Abstract Interpretation*, Rennes, France, January 2026.
8. Naïm Moussaoui Remil, Caterina Urban, and Antoine Miné. Automatic detection of vulnerable variables for CTL properties of programs. In Nikolaj S. Bjørner, Marijn Heule, and Andrei Voronkov, editors, *LPAR 2024: Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning, Port Louis, Mauritius, May 26-31, 2024*, volume 100 of *EPiC Series in Computing*, pages 116–126. EasyChair, 2024.
9. Alan Turing. Checking a Large Routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, pages 67–69, 1949.
10. C. Urban. FuncTion: An abstract domain functor for termination (competition contribution). In *Proc. TACAS*, LNCS 9035, pages 464–466. Springer, 2015.
11. Caterina Urban. *Static Analysis by Abstract Interpretation of Functional Temporal Properties of Programs (Analyse Statique par Interprétation Abstraite de Propriétés Temporelles Fonctionnelles des Programmes)*. PhD thesis, École Normale Supérieure, Paris, France, 2015.
12. Caterina Urban and Antoine Miné. A Decision Tree Abstract Domain for Proving Conditional Termination. In Markus Müller-Olm and Helmut Seidl, editors, *Static Analysis - 21st International Symposium, SAS 2014, Munich, Germany, September 11-13, 2014. Proceedings*, volume 8723 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 2014.
13. Caterina Urban and Antoine Miné. An Abstract Domain to Infer Ordinal-Valued Ranking Functions. In Zhong Shao, editor, *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8410 of *Lecture Notes in Computer Science*, pages 412–431. Springer, 2014.
14. Caterina Urban, Samuel Ueltschi, and Peter Müller. Abstract interpretation of CTL properties. In Andreas Podelski, editor, *Static Analysis - 25th International Symposium, SAS 2018, Freiburg, Germany, August 29-31, 2018, Proceedings*, volume 11002 of *Lecture Notes in Computer Science*, pages 402–422. Springer, 2018.