BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2025
http://excel.fit.vutbr.cz

Static Analysis of Heap-Manipulating Programs using Separation Logic

Tomáš Brablec*

Abstract
This work introduces a static analyzer for C programs focused on verifying the correct handling of dynamically allocated memory, specialized in programs using linked lists. The approach is based on data-flow analysis, while memory states are represented using separation logic formulae. It is implemented in the Frama-C framework. The tool was benchmarked on the linked lists subset of the SV-COMP benchmarks and compared with similar verification tools. While not reaching the performance of top competitors in this category, it is on par with most verifiers. Possible future extensions of this tool, including the integration with another analyzer, are discussed.

*xbrabl04@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Languages such as C use manual memory management, where the programmer is responsible for correctly allocating memory and handling pointers to these allocations. This carries the risk of causing one of several memory safety bugs. These include *use-after-free* – access to an already freed allocation, *double-free* – freeing an already freed allocation, or a memory leak, a loss of all references to an allocation without ever freeing it. Some of these bugs are a common cause of security vulnerabilities, so there is a need to prevent them from reaching production systems.

Several techniques are used to ensure the memory safety of programs. One approach is static analysis and formal verification. Formal verification is an umbrella term for any method that can prove the correctness of programs or a particular property. In this case, the desired property is the correctness of handling dynamically allocated memory.

One of the methods for static analysis is so-called *data-flow analysis*, which involves traversing the program's control flow graph (CFG) and tracking all possible values of variables in the program. In the context of memory safety, the values we are interested in are the targets of pointers and the data structures in the program's heap.

Formulae of *separation logic* (SL) [1, 2] are often used for this purpose, allowing us to represent data structures in dynamic memory efficiently. The *separating conjunction* operator allows us to describe disjoint parts of the heap, and *spatial predicates* describe unbounded data structures.

2. Analysis

This work implements a static analyzer capable of verifying the memory safety of C programs, with a focus on linked lists. It is an extension of the method introduced in [3]. The tool is based on data-flow analysis, and program states are represented using SL formulae. The analysis is implemented in the *Frama-C framework* [4], which provides a simplified AST of the analyzed program, libraries for AST transformation, and for implementing the data-flow analysis. The *Astral solver* [5] is used to evaluate entailments of SL formulae, which is necessary to find fixpoints for loops.

The implementation itself can be found on GitHub and consists of several parts.

2.1 Preprocessing

AST preprocessing serves multiple purposes. One purpose is simplifying complex expressions in assignments, function arguments, and conditions to make the analysis implementation easier. Another purpose

is to analyze the C structures defined in the program and determine which type of linked list they describe. This is done using a heuristic that infers the list type from the types of the fields.

2.2 Analysis of Statements

Another part of the implementation is the analysis of individual statements. When processing a statement, it is necessary to modify the formulae representing the program's state before executing the statement so that they describe the state after its execution. These statements include several kinds of dereferences and field accesses, memory allocation and deallocation, and function calls. There is a cache for the results of analyzing function calls called *function summaries*.

2.3 Simplifications of SL Formulae

Before the analysis of a new statement is started, the formulae carried from the previous statement are simplified. This includes, for example, removing unreachable spatial predicates from formulae (these are then reported as memory leaks), removing unnecessary variables from equivalences, or renaming variables at the end of their lexical scope.

2.3.1 Abstraction

The abstraction is a process of finding spatial predicates that form a chain in a formula and merging them into a list predicate that describes a linked list of arbitrary length. This is needed for the convergence of the analysis on unbounded data structures. The analysis supports three types of lists – singly and doubly linked lists, and nested lists. Both linear and cyclic lists are supported.

3. Results

3.1 Manual Testing

The analyzer was tested on a manually created set of test programs that work with all supported types of lists. The tested operations include allocating lists with nondeterministic length, iteration through the list nodes, adding and removing nodes, reversing a list, and deallocating a list. The analyzer was able to verify the correctness of all these simple programs.

When bugs were introduced into these programs, the tool was able to detect them. The detected errors include dereferencing a pointer to deallocated memory, dereferencing a pointer that may be null, memory leaks, and freeing an already freed memory.

3.2 SV-COMP

Further testing was conducted on a dataset from the SV-COMP competition, specifically on the subset of benchmarks focused on linked lists. This dataset contains a total of 134 programs testing a wide range of operations on many types of lists. These programs include, for example, sorting lists, traversal and modification of cyclic lists, or combining several types of lists in a single program.

On this dataset, the tool can correctly analyze 77 programs (full results are available in the poster). Compared to the winner of this subcategory, PredatorHP [6], this tool does not support the analysis of pointer arithmetic. Currently, it is impossible to represent numeric offsets into structures using SL formulae. This prevents the successful analysis of several dozen tests from the SV-COMP dataset.

4. Future Work

One limitation of the current method is the absence of numeric value analysis, which prevents a more accurate analysis of exit conditions in loops. Such analysis would be necessary for more precise error detection in programs.

One option is to add a simple numeric value analysis into the tool directly. Separation logic, as implemented by the Astral solver, allows for inserting SMT terms into formulae that can represent the values of numeric variables.

A more interesting approach is to integrate this analyzer into EVA [7], as both tools would benefit from exchanging information during analysis. This tool could provide EVA with more precise information about pointers, such as the validity of dereferences inside loops iterating on lists. EVA could, in turn, provide more accurate values of integer variables to analyze conditions better.

5. Conclusions

A tool for static analysis of programs focused on linked lists was implemented. The approach is based on data-flow analysis and uses separation logic to represent program states. The tool was tested on the SV-COMP benchmark and passed most expected tests.

Acknowledgements

I would like to thank my supervisor, Ing. Tomáš Dacík, and my advisor, prof. Ing. Tomáš Vojnar, Ph.D., for introducing me to the topic of software verification and for their guidance during the design and implementation of this project.

References

[1] J.C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, 2002.

[2] S. Ishtiaq and P.W. O'Hearn. Separation and Information Hiding. In *Proc. of POPL'01*. ACM, 2001.

[3] Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. A local shape analysis based on separation logic. In Holger Hermanns and Jens Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 287–302, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[4] Julien Signoles, Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, and Boris Yakobowski. Frama-c: a software analysis perspective. volume 27, 10 2012.

[5] Tomáš Dacík, Adam Rogalewicz, Tomáš Vojnar, and Florian Zuleger. Deciding boolean separation logic via small models. In Bernd Finkbeiner and Laura Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 188–206, Cham, 2024. Springer Nature Switzerland.

[6] Kamil Dudka, Petr Peringer, and Tomás Vojnar. Byte-Precise Verification of Low-Level List Manipulation. In *In Proc. of SAS*, volume 7935 of *LCNS*, pages 215–237, 2013.

[7] David Bühler. *EVA, an Evolved Value Analysis for Frama-C: structuring an abstract interpreter through value and state abstractions*. PhD thesis, University of Rennes 1, 2017.