Static Analysis of Heap-Manipulating Programs using Separation Logic

Tomáš Brablec Supervisor: Ing. Tomáš Dacík
Advisor: prof. Ing. Tomáš Vojnar Ph.D.

BRNO UNIVERSITY OF TECHNOLOGY — FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2025

Introduction

Memory safety is a property of programs that guarantees all pointer operations to be valid. Since memory safety bugs, such as *use-after-free* or *null pointer dereferences*, often lead to exploitable vulnerabilities, there is a need to verify the memory safety of programs.

This work implements a **static analyzer** that verifies the memory safety of C programs. The analyzer is focused on **linked lists** and works by doing a dataflow analysis over the program's CFG. Formulae of **separation logic** (SL) are used to represent the abstract memory states in the program.

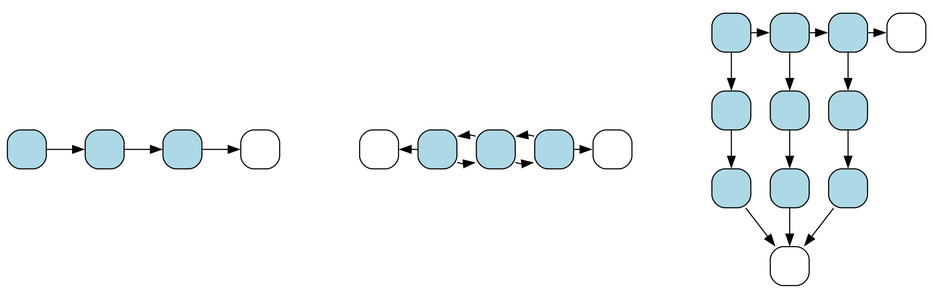

Figure 1: Supported lists types – singly linked list, doubly linked list, nested list

$\mathrm{ls}_{1+}(x, y) * y \mapsto z$

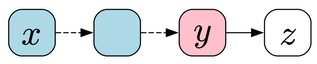

Figure 2: Example of an SL formula and its model, the formula represents a disjoint partition of the heap into a list segment and a pointer.

Analysis

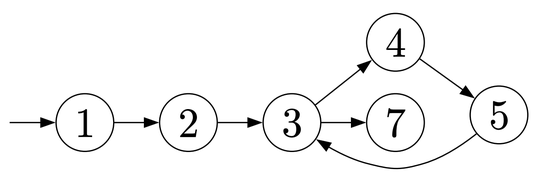

Figure 3: Control flow graph of the program in Figure 4

Code	state formulae
`1 Node *x = malloc(size);`	$x \mapsto f'_1$
`2 Node *y = x;`	$x \mapsto f'_1 * y = x$
`3 while (rand()) {`	$x \mapsto f'_1 * y = x$ $x \mapsto y * y \mapsto f'_2$ $\mathrm{ls}_{2+}(x, y) * y \mapsto f'_3$
`4  y->next = malloc(size);`	$x \mapsto f'_1 * f'_1 \mapsto f'_2 * y = x$ $x \mapsto y * y \mapsto f'_2 * f'_2 \mapsto f'_3$ $\mathrm{ls}_{2+}(x, y) * y \mapsto f'_3 * f'_3 \mapsto f'_4$
`5  y = y->next;`	$x \mapsto y * y \mapsto f'_2$ $x \mapsto f'_4 * f'_4 \mapsto y * y \mapsto f'_3$ $\mathrm{ls}_{2+}(x, f'_5) * f'_5 \mapsto y * y \mapsto f'_4$
`6 }`	
`7 y->next = NULL;`	$\mathrm{ls}_{0+}(x, y) * y \mapsto \bot$

Figure 4: Simplified progress of an analysis run, (f'_n represents an existentially quantified variable)

- before loop + first loop iteration
- second loop iteration
- third loop iteration + after loop

$$\bigvee \begin{bmatrix} x \mapsto f'_1 * y = x \\ x \mapsto y * y \mapsto f'_2 \\ \mathrm{ls}_{2+}(x, y) * y \mapsto f'_3 \end{bmatrix} \quad \models \quad \exists f'_3.\ \mathrm{ls}_{0+}(x, y) * y \mapsto f'_3$$

Figure 5: Entailments are verified by the Astral solver, allowing the analysis to find fixpoints for loops

Technologies

Software Analyzers

- OCaml – the analyzer is implemented using functional programming
- **Astral solver** [1] – solver for separation logic, used for evaluating entailments of SL formulae in the join operation
- **Frama-C framework** [2] – provides a library for dataflow analysis, AST representation of the analyzed program, utilities for AST manipulation, and a GUI (Ivette) for interactive analysis

Capabilities

- Support for both linear and circular list variants
- Analysis across function calls with **function summaries**
- Pointers to variables on the stack are supported
- Some integer bounds on loops are supported through loop unrolling

Results

The analyzer was tested on the SV-COMP benchmarks, specifically on the Linked Lists subset. From the total of **134 tests, 77 are passing**. Compared to the 2024 winner in this category, PredatorHP, this analyzer cannot analyze pointer arithmetic.

Tests (134 total)	This work	PredatorHP	EVA
Correct	77	124	80
Correct (true)	73	96	76
Correct (false)	4	28	4
Incorrect	0	0	40

However, this tool correctly analyzes 15 tests, for which the EVA analyzer built into Frama-C produces wrong results, and unlike EVA, it can find memory leaks. It could, therefore, be worth it to **integrate our analysis method into EVA** itself, improving the precision of both tools by exchanging information during analysis.

Manual testing shows that some simple **practical programs can be verified** with the current version of the tool, such as a postfix expression calculator that uses a linked list implementation of a stack.

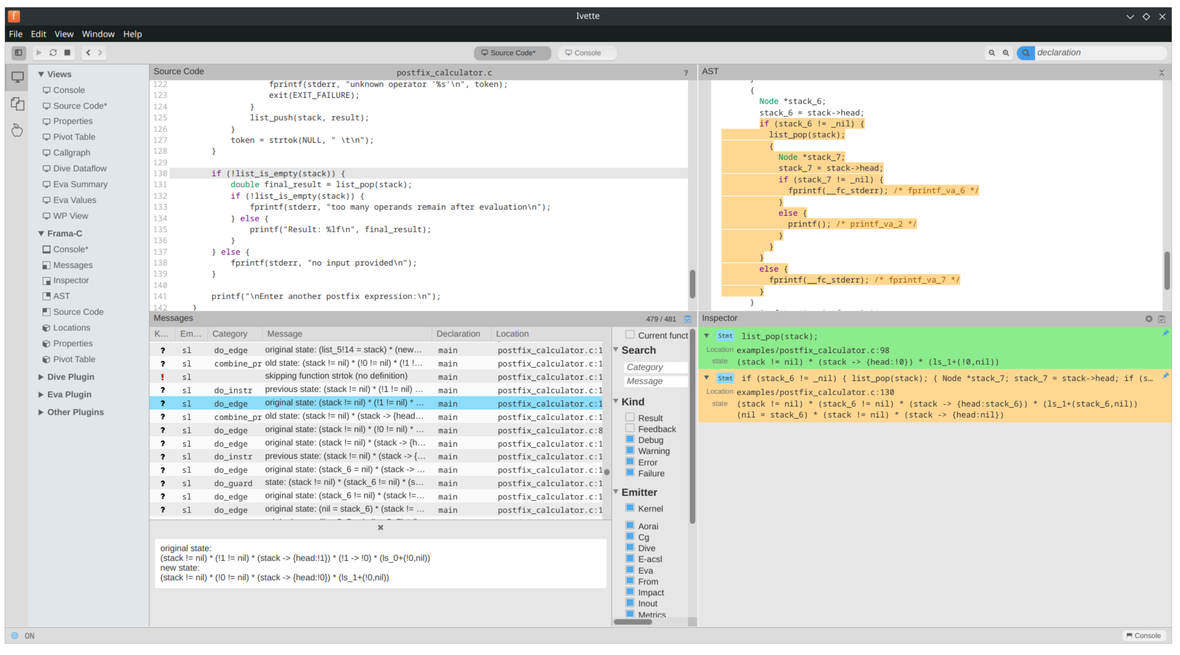

Figure 6: Ivette showing the analysis results

Future Work

- Integrate with analyzers for other domains (e.g. EVA)
- Detect types of lists dynamically from structures in the heap
- Improve the analysis of non-pointer conditions to improve bug detection
- Implement the remaining C language features (such as global variables)

[1] T. Dacík et al., "Deciding Boolean Separation Logic via Small Models," in *TACAS*, 2024.

[2] J. Signoles et al., "Frama-C: a Software Analysis Perspective," in *Formal Aspects of Computing*, 2015.