Static Analysis of Heap-Manipulating Programs using Separation Logic

Author
Tomáš Brablec

Supervisor
Ing. Tomáš Dacík

Advisor
prof. Ing. Tomáš Vojnar, Ph.D.

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Motivation

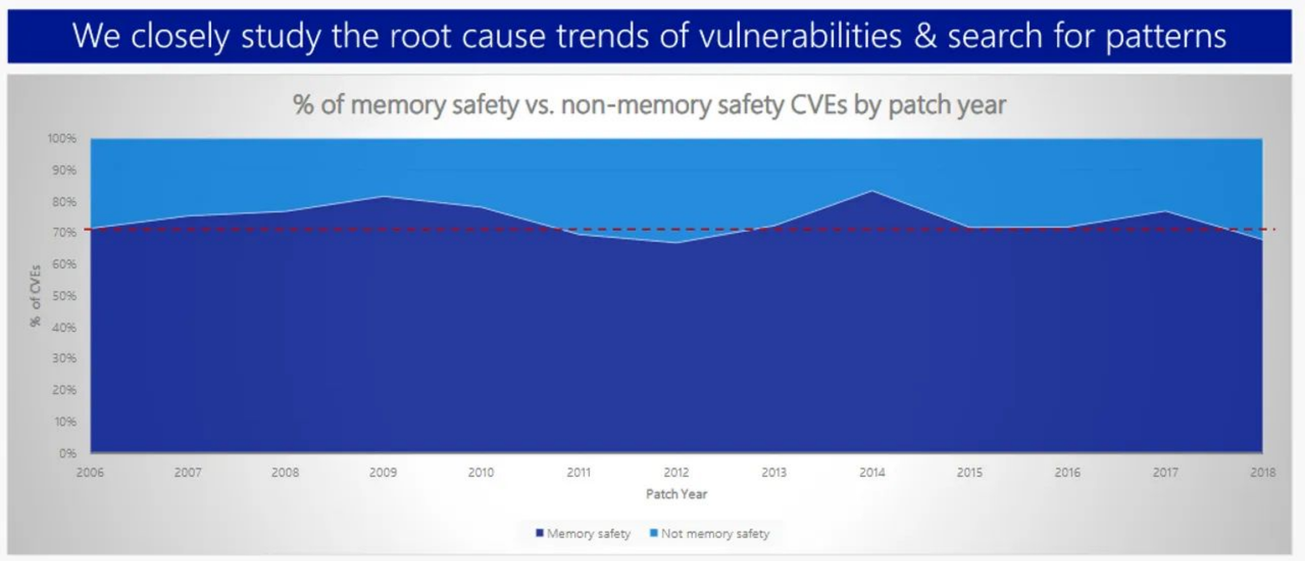
- Manual memory management creates a whole class of bugs
- Use-after-free, double-free, memory leaks, …
- Memory safety errors are a common source of security vulnerabilities
- Linked lists are a common data structure in low level software

CVE-2024-12382 Detail

Description

Use after free in Translate in Google Chrome prior to 131.0.6778.139 allowed a remote attacker to potentially exploit heap corruption via a crafted HTML page. (Chromium security severity: High)

Example of a recent memory safety bug



Microsoft: Around 70% of CVEs are memory related

Verification Tool

- This work introduces the KTSN verification tool
- Goals: verification of memory safety and bug detection
- Focused on programs with variants of linked lists
- Abstract memory states are encoded in separation logic (SL)
- Compared to existing approaches, a dedicated solver Astral is used for checking SL formulae:
    - Better modularity
    - Higher precision than syntactic comparison
- Implemented in the Frama-C framework using dataflow analysis

Astral

Software Analyzers

Implementation

- Analyzed program's AST is preprocessed into simple instructions
- These instructions are interpreted as changes to the SL formulae
- Invariants for loops are found using abstraction and checking entailments of formulae using the solver
- Analyses of function calls are cached using function summaries

Code	State formulae
`Node *x = malloc(size);`	$x \mapsto f'_1$
`Node *y = x;`	$x \mapsto f'_1 * y = x$
`while (rand()) {`	$x \mapsto f'_1 * y = x$ $x \mapsto y * y \mapsto f'_2$ $\mathtt{ls}_{2+}(x, y) * y \mapsto f'_3$
`  y->next = malloc(size);`	$x \mapsto f'_1 * f'_1 \mapsto f'_2 * y = x$ $x \mapsto y * y \mapsto f'_2 * f'_2 \mapsto f'_3$ $\mathtt{ls}_{2+}(x, y) * y \mapsto f'_3 * f'_3 \mapsto f'_4$
`  y = y->next;`	$x \mapsto y * y \mapsto f'_2$ $x \mapsto f'_4 * f'_4 \mapsto y * y \mapsto f'_3$ $\mathtt{ls}_{2+}(x, f'_5) * f'_5 \mapsto y * y \mapsto f'_4$
`}`	$x \mapsto f'_1 * y = x$ $\mathtt{ls}_{1+}(x, y) * y \mapsto f'_2$
`y->next = NULL;`	$x \mapsto \mathtt{nil} * y = x$ $\mathtt{ls}_{1+}(x, y) * y \mapsto \mathtt{nil}$

simplified analysis progress

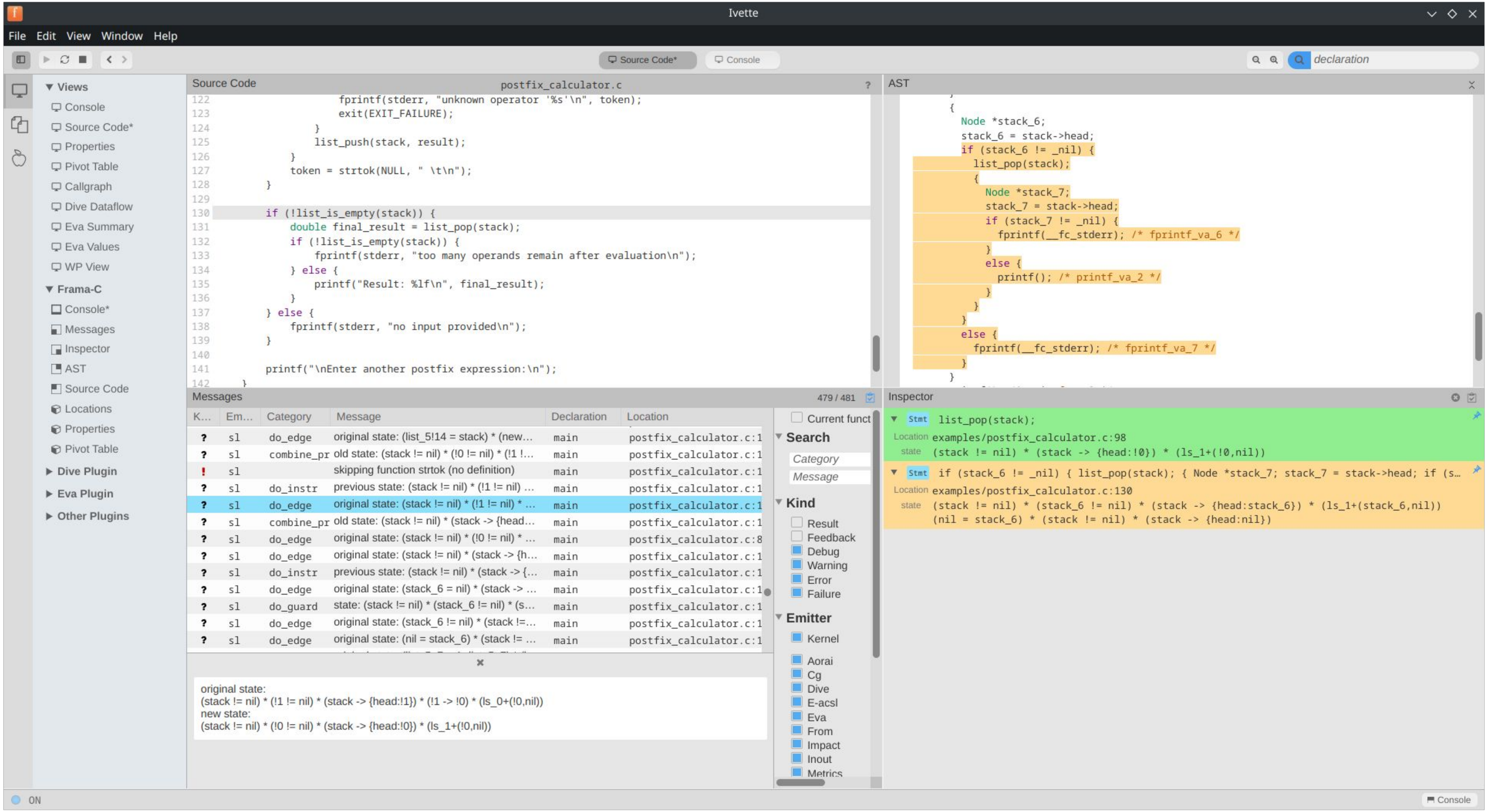
Results

- Tested on crafted programs:
  - Verification of programs with singly/doubly linked lists and nested lists
  - Can find use-after-free and double-free bugs, null-pointer dereferences, and memory leaks
- Tested on SV-COMP benchmarks, surpasses all but two other tools on linked lists
- Outperforms the EVA analyzer from Frama-C on most benchmarks in this category
- Presented the results at Excel@FIT

Analyser	Score	Safe	Unsafe
PredatorHP	220	96	28
CPAchecker	208	90	28
KTSN	154	74	6
symbiotic	130	51	28
CBMC	127	50	28
Bubaak	114	48	18
Mopsa	96	48	0
ESBMC	96	50	28
2LS	90	46	14
DIVINE	84	42	0
UAutomizer	22	10	2
sv-sanitizers	17	0	17
UTaipan	14	6	2

Comparison with SV-COMP'25 participants

Results in Ivette (Frama-C GUI)



Conclusion

- Implemented a static analyzer able to verify the memory safety of programs working with linked lists
- The method outperforms most other analyzers in this category, while being more flexible

Future work

- Compete with other analyzers in SV-COMP 2026
- Extend the analysis to other types of lists
- Integrate our method into the EVA analyzer to improve its analysis of programs with linked lists

Interpretation of Results

“Co znamenají řádky "Correct (true)", "Correct (false)", "Incorrect (true)" a "Incorrect (false)" v tabulce 1?”

Table 1: The results of evaluating different analyzers on the SV-COMP dataset

Tests (134 total)	**KTSN**	**PredatorHP**	**EVA**
Correct	80	124	56
Correct (true)	74	96	50
Correct (false)	6	28	6
Incorrect (true)	0	0	6
Incorrect (false)	0	0	48
Timeout	11	10	4
Unknown	43	0	20

- Correct (true) - Analyzer correctly verified 74 programs
- Correct (false) - Analyzer correctly identified 6 programs with a bug
- Incorrect (true) - Analyzer did not report any false positives
- Incorrect (false) - Analyzer did not report any false negatives

Supported list types

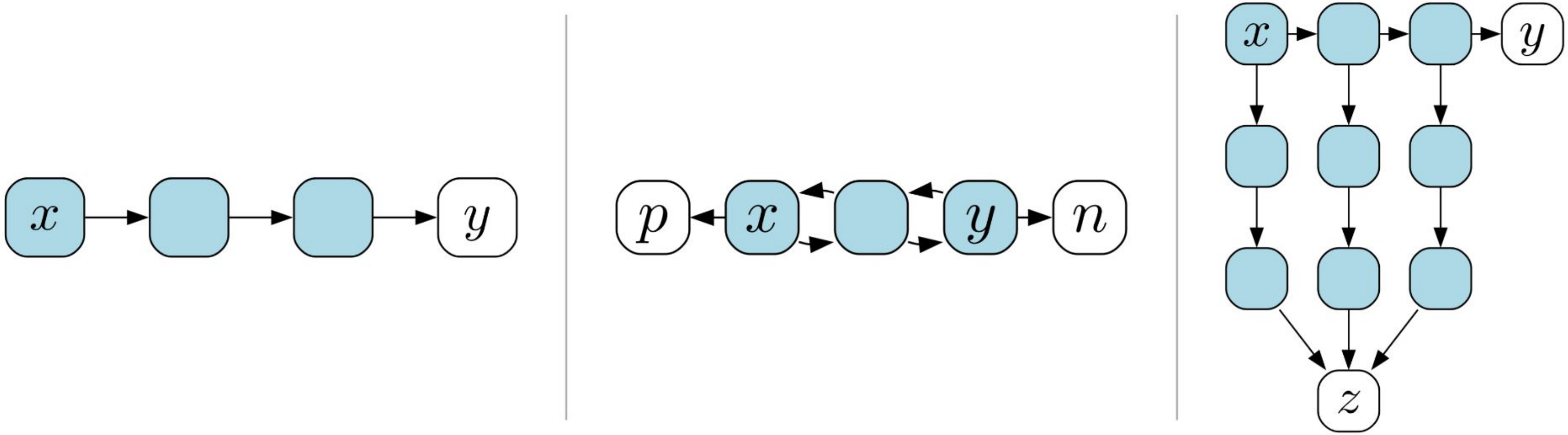
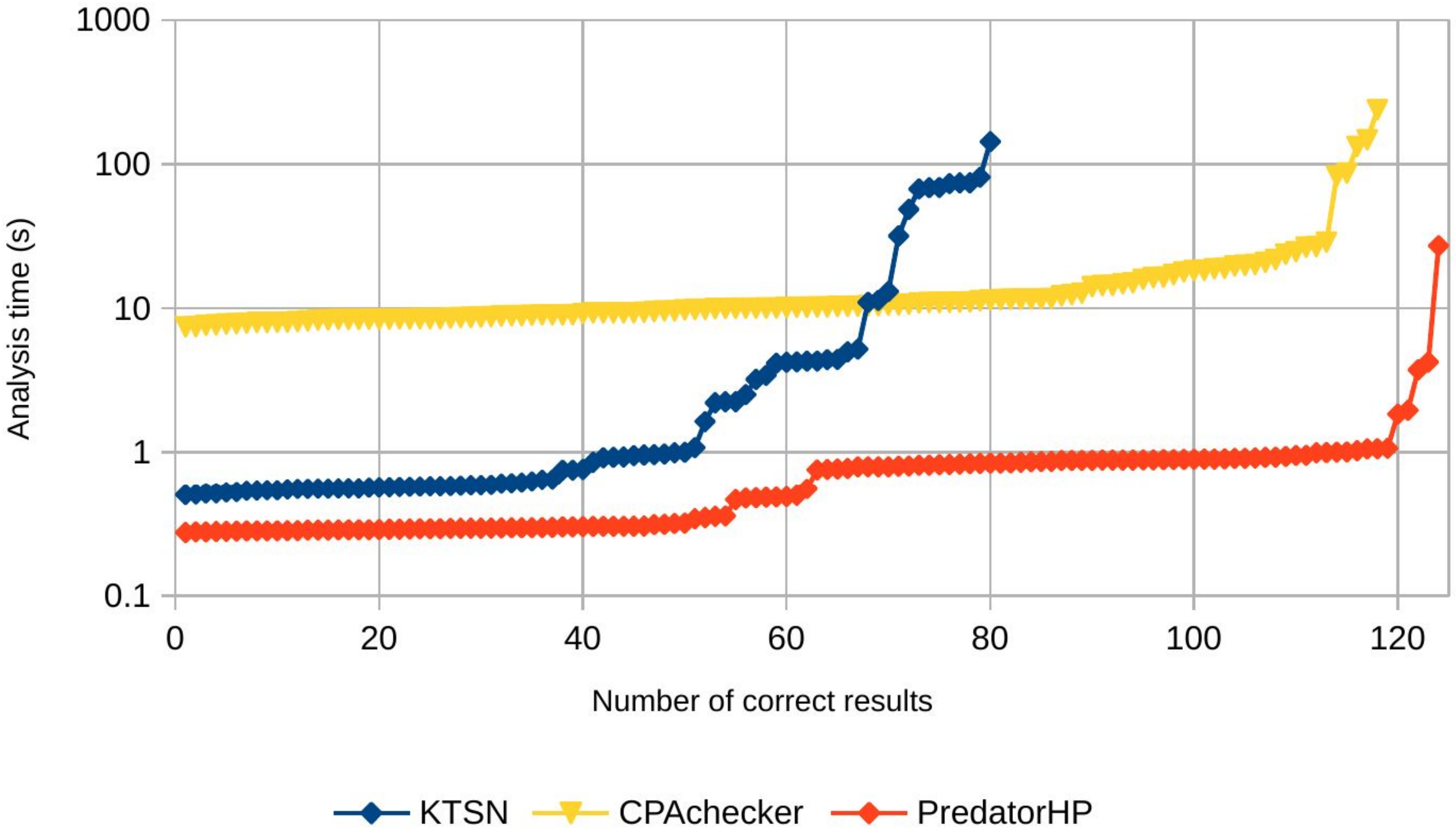


Figure 1: Supported types of lists with the allocated nodes colored blue: singly-linked list, doubly-linked list, nested list

Comparison of analysis times



Time spent in KTSN and Astral

FIT

Table 3: Examples of the fastest and slowest benchmarks and the time spent in the solver versus in the analyzer itself

Benchmark	**Total time (s)**	**Astral time (s)**	**Analyzer time (s)**	**Number of queries**
`sll2c_append_unequal.c`	0.50	0.00	0.50	2
`sll_shallow_copy-1.c`	0.51	0.00	0.51	2
`dll2c_prepend_equal.c`	0.51	0.02	0.49	4
		...		
`dll-rb-cnstr_1-2.c`	16.29	15.64	0.65	141
`sll-sorted-2.c`	31.79	31.65	0.14	1472
`dll-simple-white-blue-2.c`	48.61	48.13	0.48	816