Current State of the MemorySafety Analysis in CPAchecker

Daniel Baier

September 9, 2024

LMU Munich, SoSy-Lab

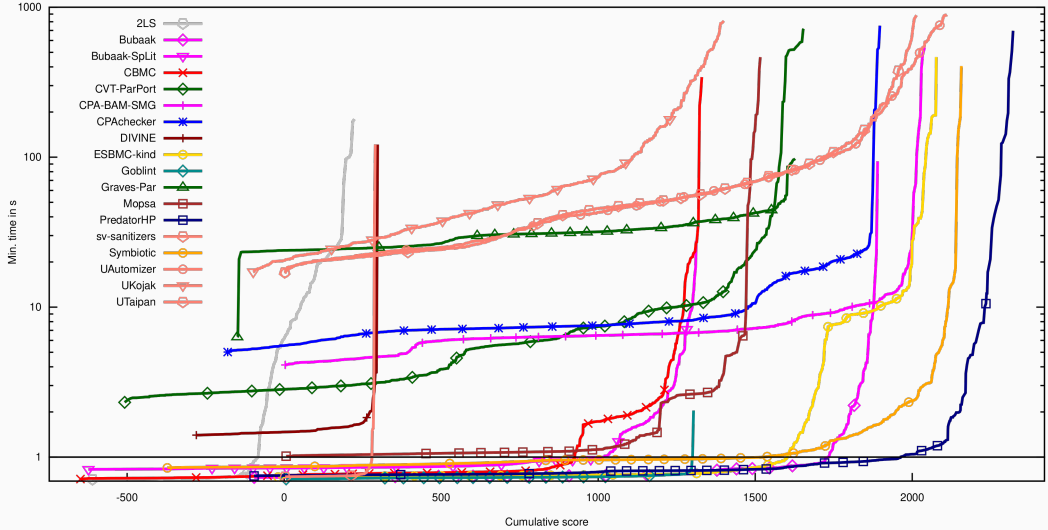
MemorySafety Analysis in CPAchecker [2]

- Graph based memory model based on Symbolic Memory Graphs [4]
- Utilizes Symbolic Execution [3]
- Detects invalid pointer access (invalid-deref)
- Detects invalid freeing of memory (invalid-free)
- Detects memory leaks (valid-memtrack/memcleanup)
- Specializes in list-abstraction

MemorySafety Analysis in CPAchecker

- Superseded the old MemorySafety analysis a year ago
- Was used in SV-COMP24 [1] with moderate success

MemorySafety Analysis in SV-COMP24



MemorySafety Analysis in CPAchecker

- Superseded the old MemorySafety analysis a year ago
- Was used in SV-COMP24 [1] with moderate success

→ However, PredatorHP [6] won

→ Our analysis is based on Predator [5]

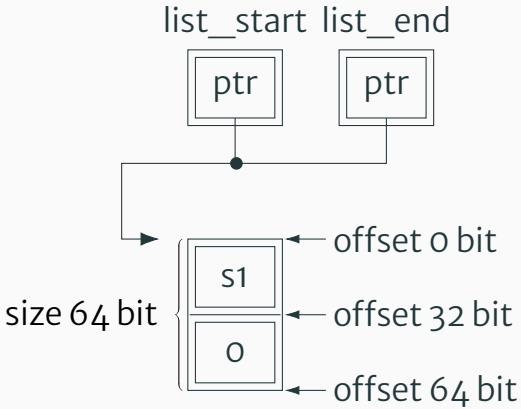
→ We have room to improve

Improvements Since Last Year

- Overall improvements
- Symbolic memory size
- Values can have symbolic offsets
- Merge operator

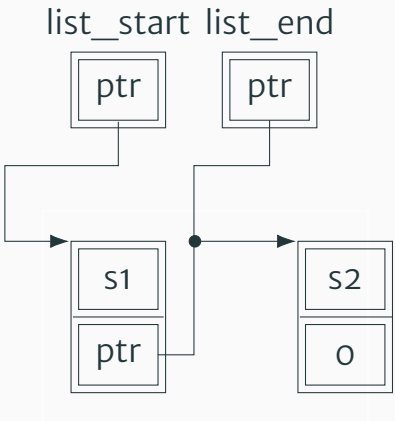
Example: Symbolic Memory Graphs

```
struct node {
int value;
struct node * next;
} List;

int main() {
List * list_start = malloc(...);
List * list_end = list_start;
list_end->value = __VERIFIER_nondet_int(); // s1
list_end->next = 0;

}
```

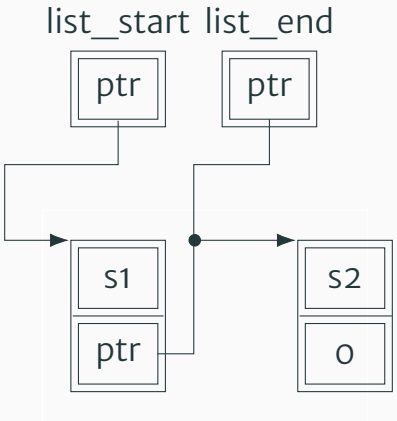


Example: Symbolic Memory Graphs

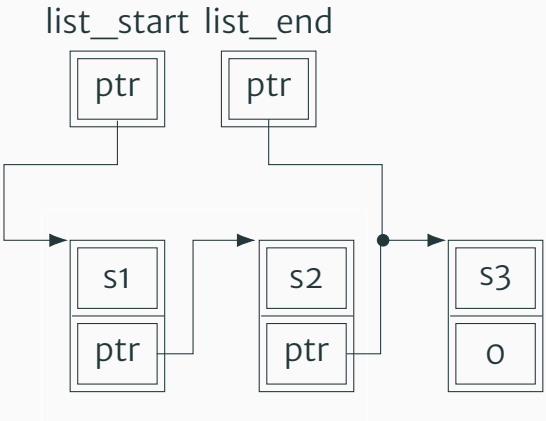
```
struct node {
int value;
struct node * next;
} List;

int main() {
List * list_start = malloc(...);
List * list_end = list_start;
list_end->value = __VERIFIER_nondet_int(); // s1
list_end->next = 0;


list_end->next = malloc(...);
list_end = list_end -> next;
list_end->value = __VERIFIER_nondet_int(); // s2
list_end->next = 0;

}
```



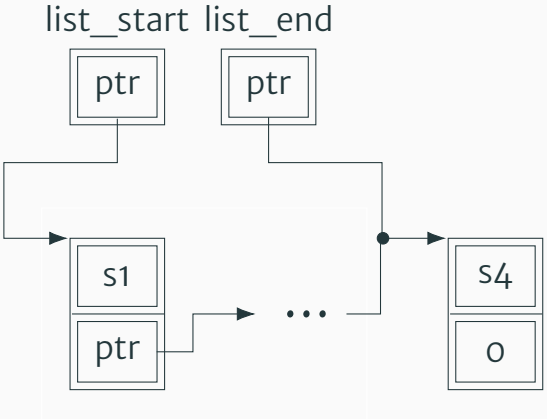
Example: Symbolic Memory Graphs – List Abstraction



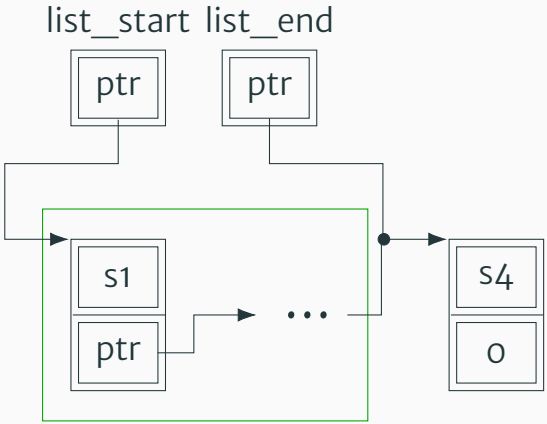
Example: Symbolic Memory Graphs – List Abstraction



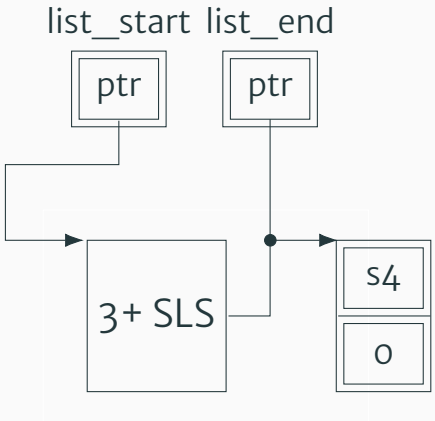
Example: Symbolic Memory Graphs – List Abstraction



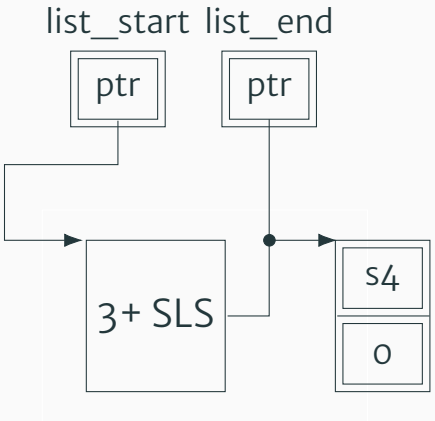
Example: Symbolic Memory Graphs – List Abstraction



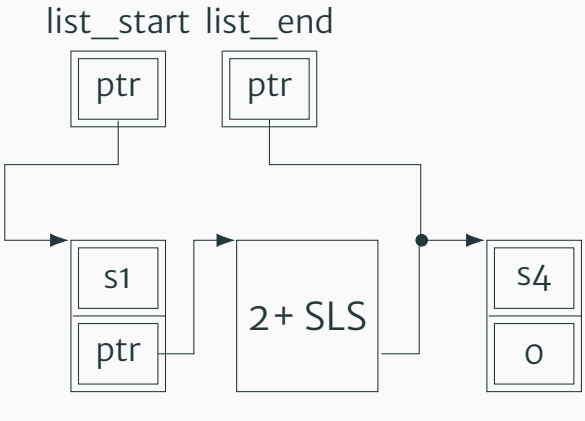
Example: Symbolic Memory Graphs – List Abstraction



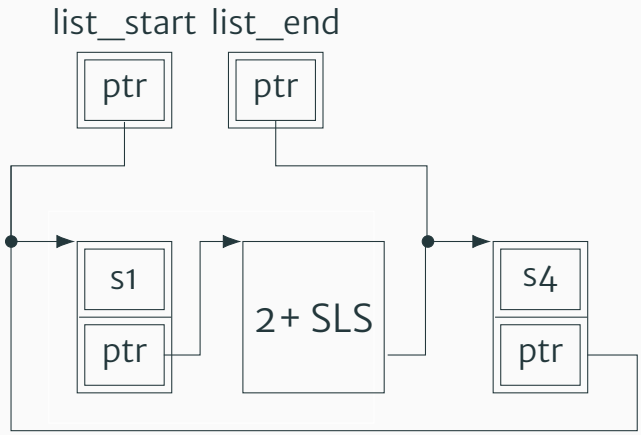
Example: Symbolic Memory Graphs – Materialization



Example: Symbolic Memory Graphs – Materialization

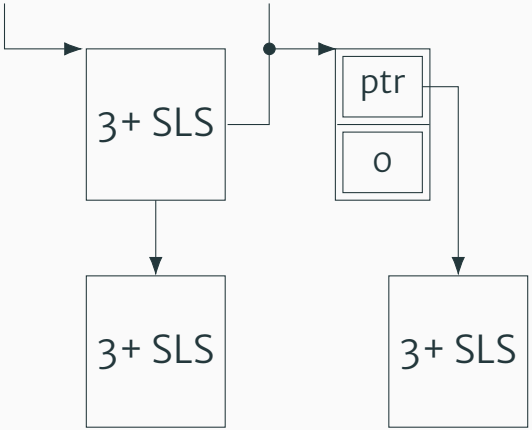


Example: Symbolic Memory Graphs – More Cases



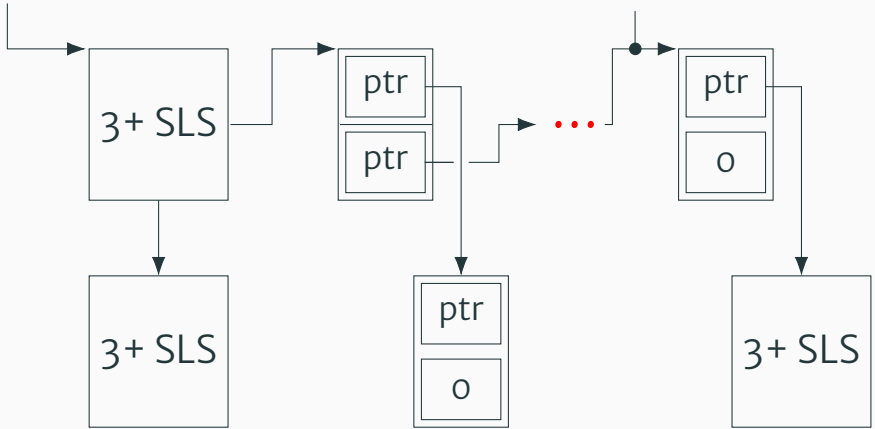
Example: Symbolic Memory Graphs – Nested Lists

Suppose we switch the symbolic values with nested lists, we would expect the list to look like this:



Example: Symbolic Memory Graphs – Nested Lists Problem

However, there is an indefinite amount of
non-abstractable list elements in between abstractable elements:

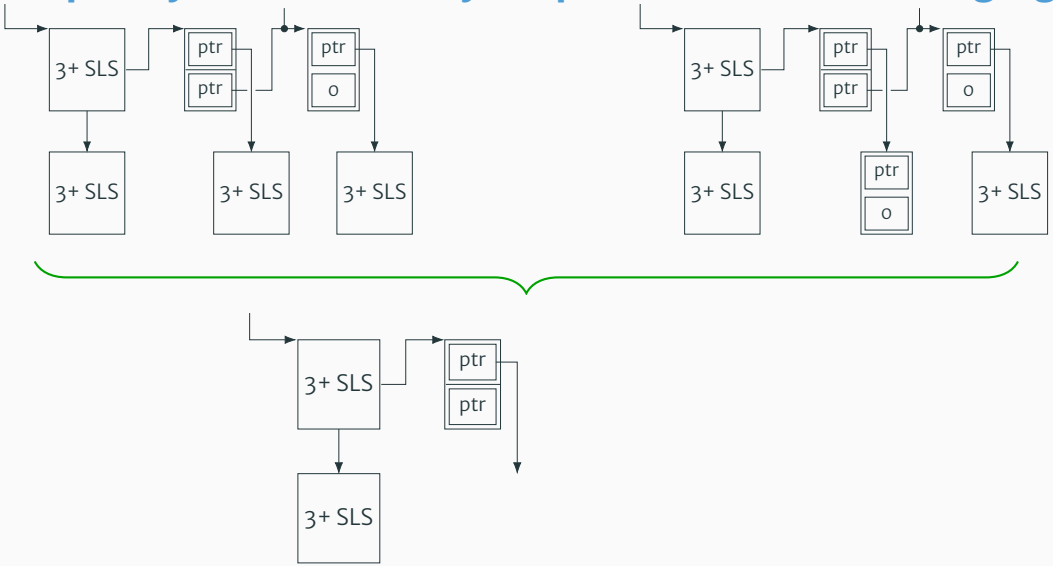


Example: Symbolic Memory Graphs – Nested Lists Solution

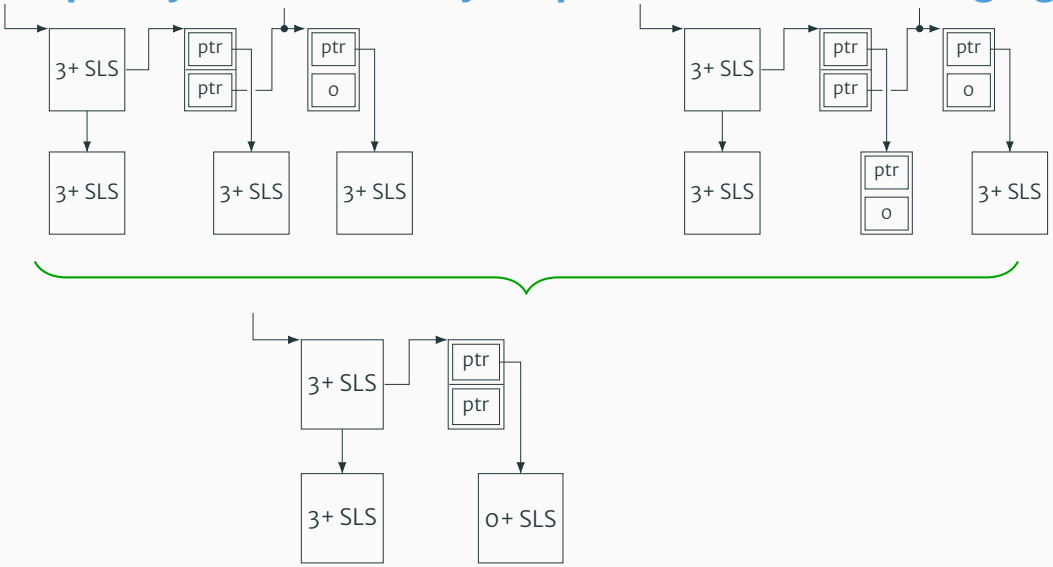
Solution: Merge

- When merging a concrete and an abstracted object, a minimal abstraction is inserted (0+)
- Leads to loss of precision
- But allows the analysis to handle more cases

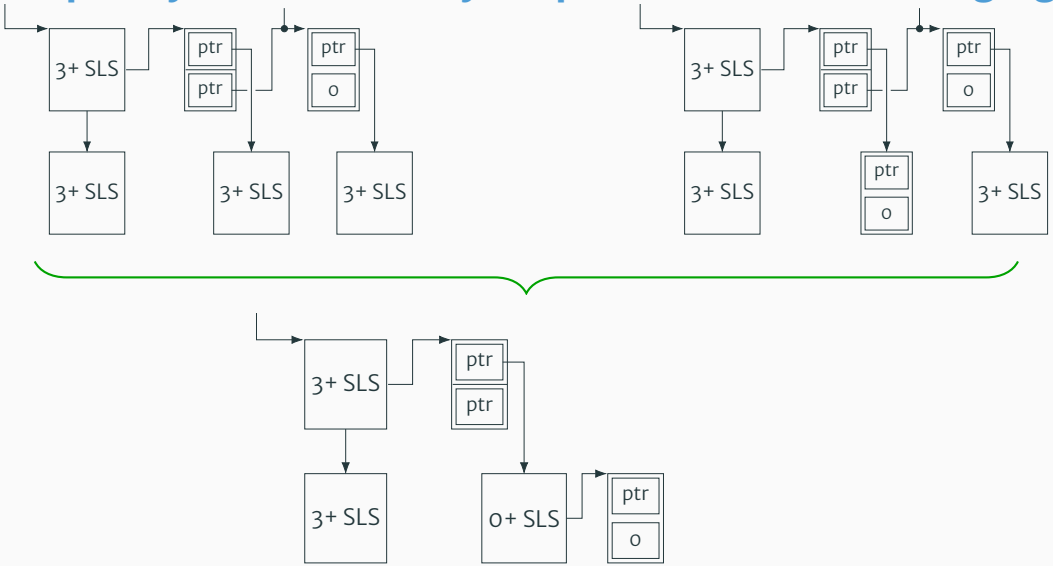
Example: Symbolic Memory Graphs – Nested Lists Merging



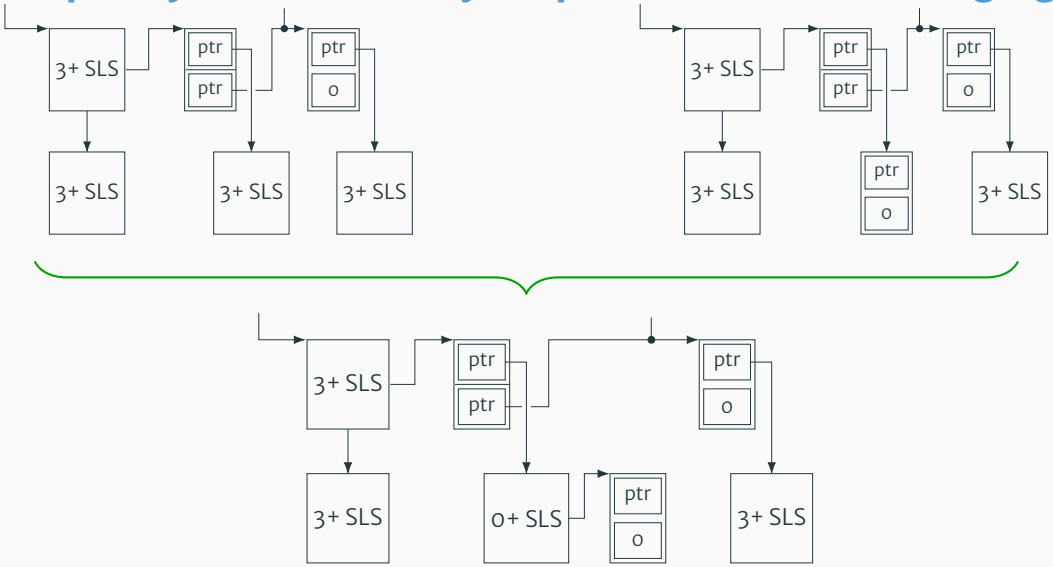
Example: Symbolic Memory Graphs – Nested Lists Merging



Example: Symbolic Memory Graphs – Nested Lists Merging



Example: Symbolic Memory Graphs – Nested Lists Merging

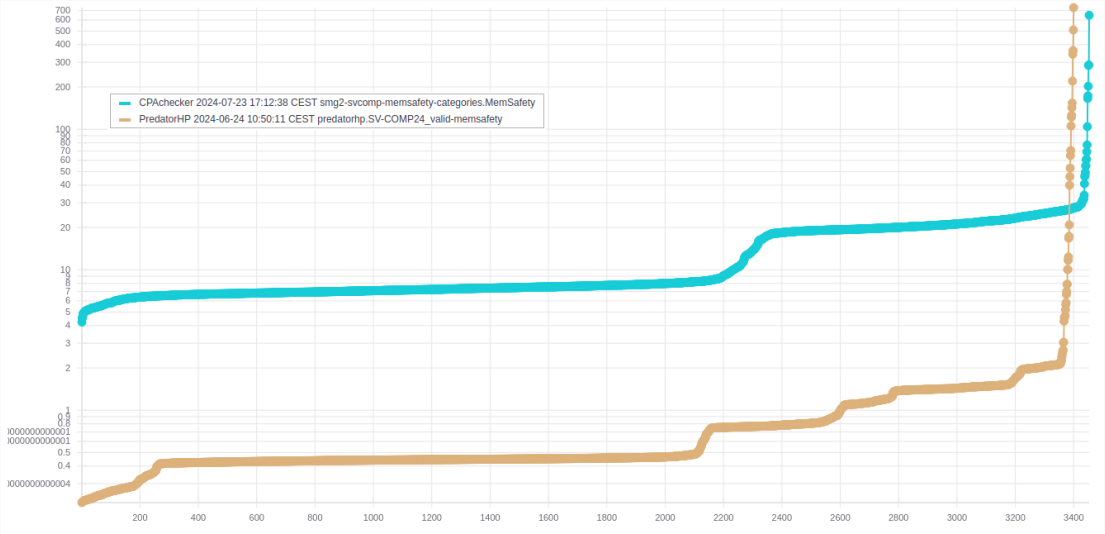


Current Results vs. PredatorHP

	CPAchecker			PredatorHP		
	status	cputime (s)	memory (MB)	status	cputime (s)	memory (MB)
all results	3773	181000	1170000	3773	**49300**	**213000**
correct results	**3453**	43300	984000	3400	**5690**	**145000**
correct true	1650	20800	435000	**1683**	**3700**	**73200**
correct false	**1803**	22500	549000	1717	**1990**	**72100**
incorrect results	**0**	–	–	4	4.87	126
incorrect true	**0**	–	–	1	.455	38.1
incorrect false	**0**	–	–	3	4.41	87.5

* Run-set used was SV-COMP24 + fixed Juliet benchmarks

Current Results vs. PredatorHP – Quantile Plot



Future Work

- Merge not yet finished completely
- Array abstraction (FunArray)
- Improve abstraction precision (e.g. via predicates or summaries)

References i

[1] Beyer, D.: State of the art in software verification and witness validation: Sv-comp 2024. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 299–329. Springer Nature Switzerland (2024). https://doi.org/10.1007/978-3-031-57256-2_15

[2] Beyer, D., Keremoglu, M.E.: CPACHECKER: A tool for configurable software verification. In: Proc. CAV. pp. 184–190. LNCS 6806, Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_16

References ii

[3] Beyer, D., Lemberger, T.: CPA-SymExec: Efficient symbolic execution in CPAchecker. In: Proc. ASE. pp. 900–903. ACM (2018). https://doi.org/10.1145/3238147.3240478, https://www.sosy-lab.org/research/pub/2018-ASE.CPA-SymExec_Efficient_Symbolic_Execution_in_CPAchecker.pdf

[4] Dudka, K., Peringer, P., Vojnar, T.: Byte-precise verification of low-level list manipulation. In: Proc. SAS. pp. 215–237. LNCS 7935, Springer (2013). https://doi.org/10.1007/978-3-642-38856-9_13

References iii

[5] Holík, L., Kotoun, M., Peringer, P., Šoková, V., Trtík, M., Vojnar, T.: PREDATOR shape analysis tool suite. In: Hardware and Software: Verification and Testing. pp. 202–209. LNCS 10028, Springer (2016). https://doi.org/10.1007/978-3-319-49052-6

[6] Müller, P., Peringer, P., Vojnar, T.: PREDATOR hunting party (competition contribution). In: Proc. TACAS. pp. 443–446. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_40