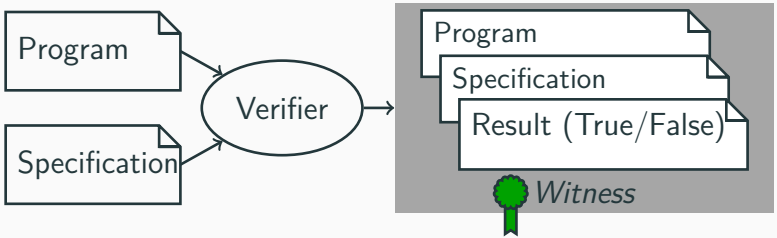
Towards
Scalable and Distributed Software Verification

Dirk Beyer, Matthias Kettl, Thomas Lemberger

LMU Munich, Germany

AVM 2024
Freiburg
2024-09-04

Automatic Software Verification



Mostly context-sensitive, whole-program analysis

Motivation

- Context: (Automatic) Software Model Checking
- We need low response time.
- Therefore, we need massively parallel approaches.
- Solution: Decomposition into blocks, construct contracts automatically
- Goal: Scalable and Distributed Software Verification

Solution: Distributed Summary Synthesis

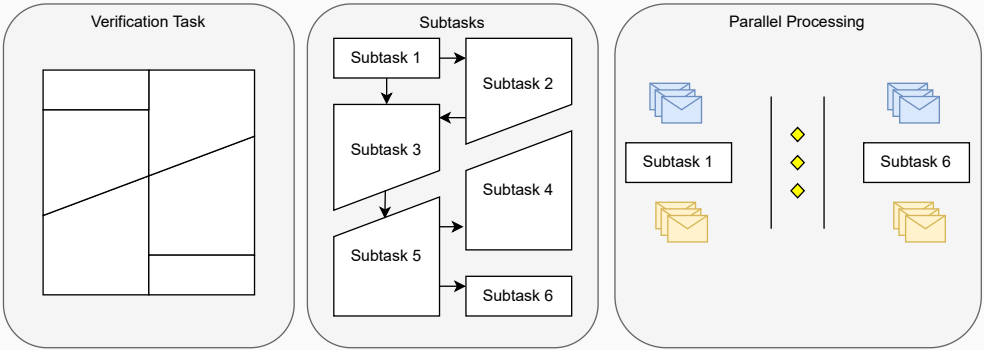
Based on [5]:

Dirk Beyer, Matthias Kettl, Thomas Lemberger:

Decomposing Software Verification using Distributed Summary Synthesis

Proc. ACM on Software Engineering, Volume 1, Issue FSE, 2024.
https://doi.org/10.1145/3660766

Overview of Decomposition

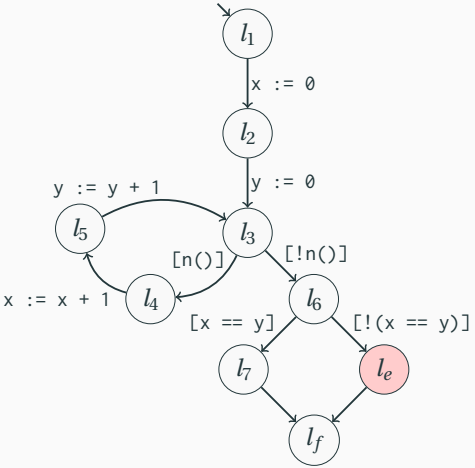


Overview of the *DSS* approach

Example: Control-Flow Automaton

```
1 int main() {
2   int x = 0;
3   int y = 0;
4   while (n()) {
5     x++;
6     y++;
7   }
8   assert(x == y);
9 }
```

Safe program



CFA of program

Decomposition

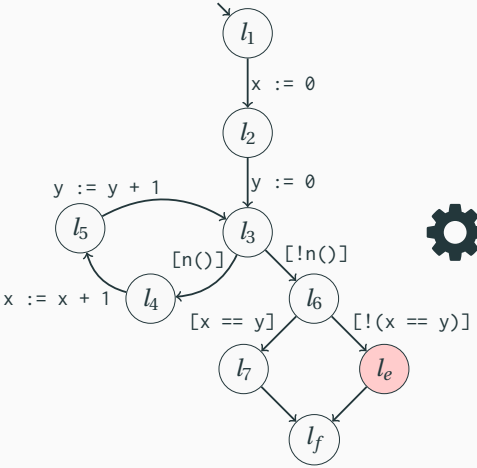
We split a large verification task into multiple smaller subtasks.

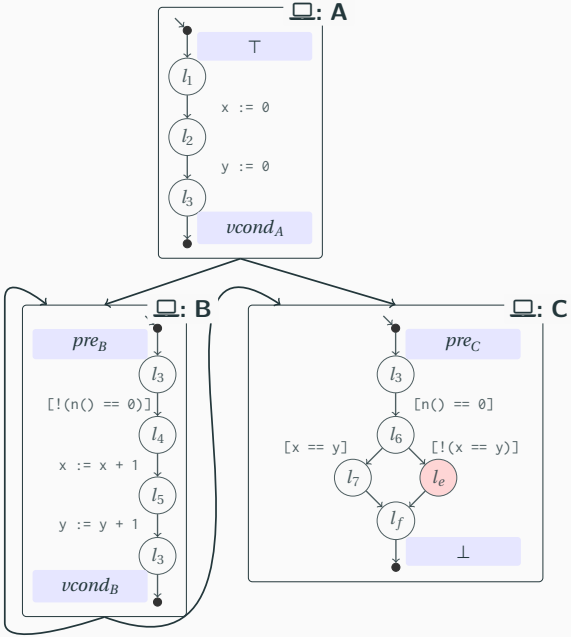
Requirements for eligible decompositions:

- Each block has exactly one entry and one exit location.
- Loops should be reflected as loops in the block graph.
- Blocks should as large as possible.
- Blocks not bound to functions.

Approach: We decompose the CFA similar to large-block encoding [3].

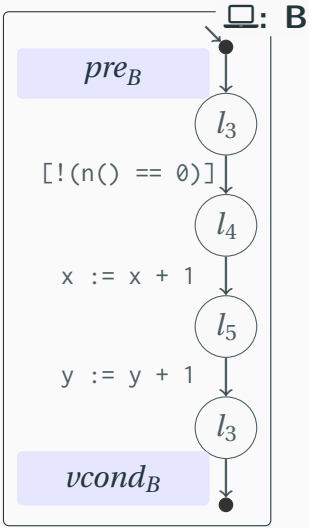
Example: Decomposition





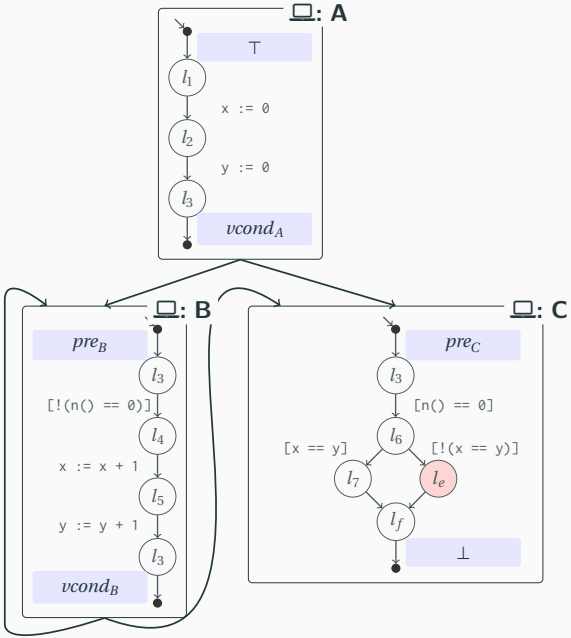
Workers

- Each worker runs independently in an own compute thread/node.
- Preconditions describe good entry states of a block (over-approximating).
- Violation condition needs to be refuted to prove a program safe.
- Preconditions are refined until all violation conditions are refuted or at least one is confirmed.



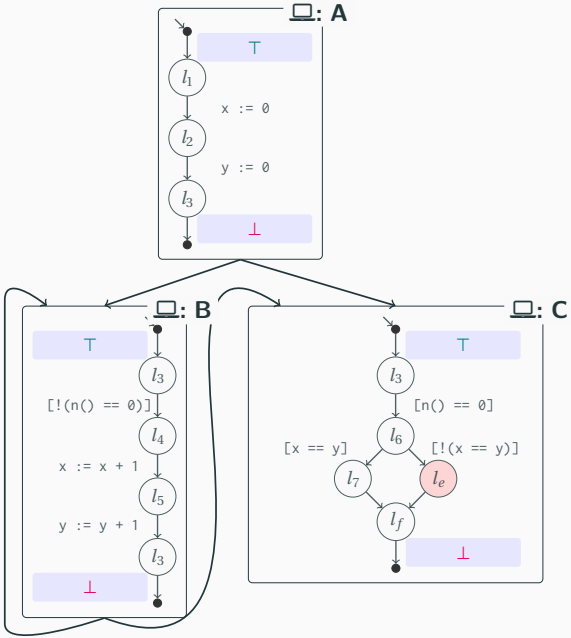
Communication Model

- Workers know their successor and predecessors.
- Workers maintain a list of preconditions, violation conditions, and their subtask.



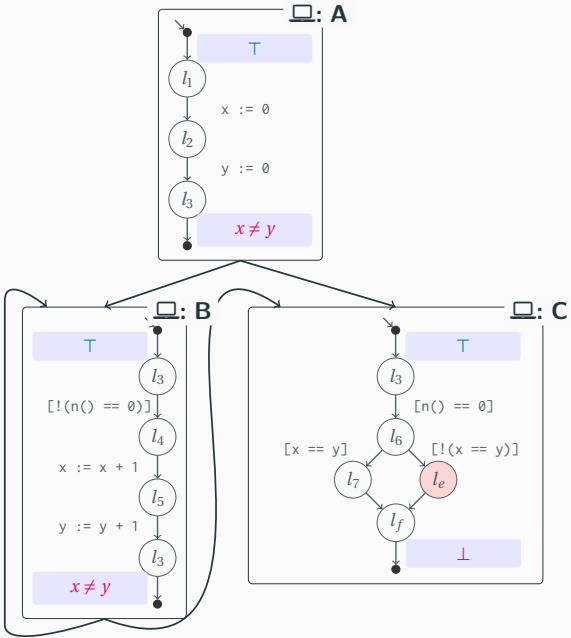
Verification with *DSS* 1

Block	Result
A	$\downarrow\boxtimes_{B,C} : \top$
B	$\downarrow\boxtimes_{B,C} : \top$
C	$\uparrow\boxtimes_{A,B} : x \neq y$



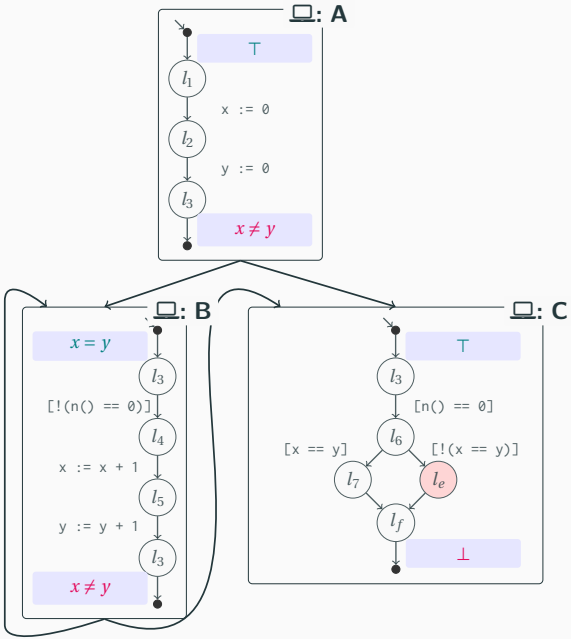
Verification with *DSS* 2

Block	Result
A	$\downarrow \boxtimes_{B,C} : x = y$
B	$\uparrow \boxtimes_{A,B} : x \neq y$
C	*idle*



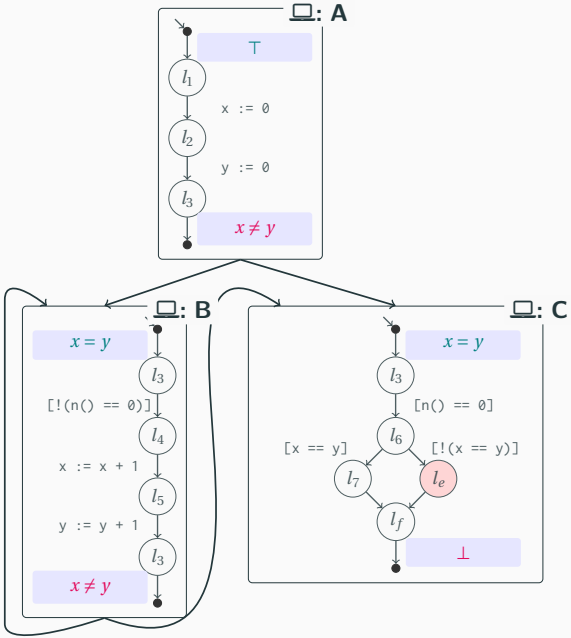
Verification with *DSS* 3

Block	Result
A	$\downarrow\!\boxtimes_{B,C} : x = y$
B	$\downarrow\!\boxtimes_{B,C} : x = y$
C	*idle*



Verification with *DSS* 4

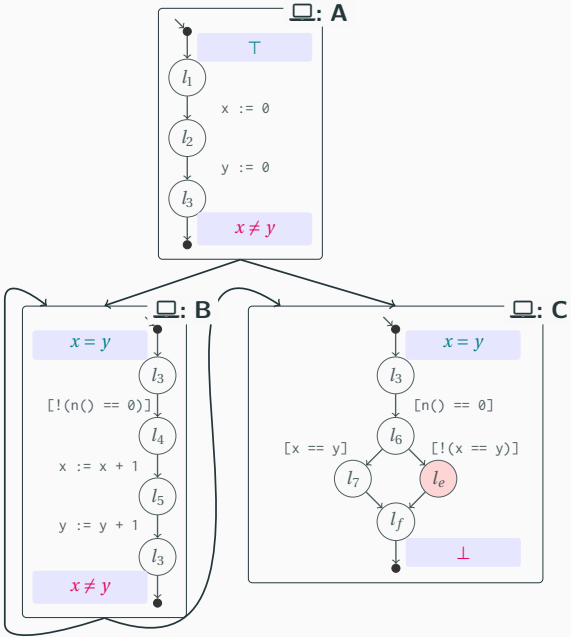
Block	Result
A	*idle*
B	*idle*
C	$\downarrow\boxtimes_{\emptyset} : \top$



Verification with *DSS* 5

Block	Result
A	*idle*
B	*idle*
C	*idle*

⇒ Fix-point reached, program safe.

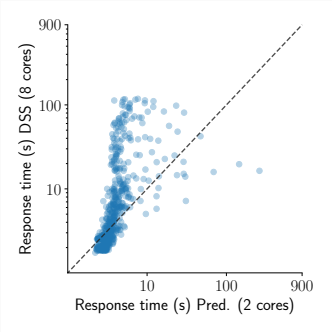


Evaluation: Setup

Benchmark Setup:

- We evaluate *DSS* on the subcategory *SoftwareSystems* of the SV-COMP '23 benchmarks.
- We focus on the 2485 safe verification tasks.
- We use the SV-COMP [2] benchmark setup:
  15 GB RAM and an 8 core Intel Xeon E3-1230 v5 with 3.40 GHz.

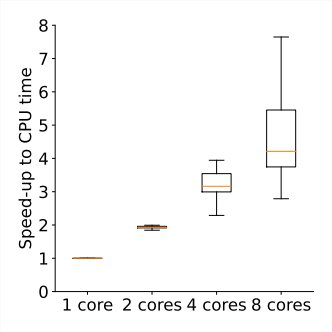
Evaluation: Results



Response time of predicate abstraction (x-axis) vs. *DSS* (y-axis).

DSS introduces overhead which only pays-off for more complex tasks.
A parallel portfolio combines the best of both worlds.

Evaluation: Distribution of Workload



The ratio of the CPU time compared to the response time for 1, 2, 4, and 8 cores.

The workload is distributed effectively to multiple processing units.

Evaluation: Outperforming Predicate Analysis

Task	$\mathrm{CPU}_{\mathbb{P}}(s)$	$\mathrm{CPU}_{DSS}(s)$	$\mathrm{RT}_{\mathbb{P}}(s)$	$\mathbf{RT}_{DSS}(s)$	# threads
leds–leds-regulator...	44.8	33.2	30.8	7.18	92
rtc–rtc-ds1553.ko-l...	49.0	64.6	30.3	14.0	164
rtc–rtc-stk17ta8.ko...	46.7	67.9	28.9	15.1	162
watchdog–it8712f_w...	86.8	50.3	69.0	15.9	216
ldv-commit-tester/m0...	50.1	103	28.8	21.0	230

DSS introduces overhead which only pays-off for more complex tasks.
A parallel portfolio combines the best of both worlds.

Related Approaches

Existing approaches have limitations that distributed summary synthesis solves, most importantly the potential to scale to many nodes:

- Infer [6, 7] scales well but reports many false alarms.
  - ⇒ *DSS* inherits all properties of the underlying analysis.
- BAM [4] has nested blocks that are not parallelizable.
  - ⇒ *DSS* parallelizes as much as possible.
- HiFrog [1] is bound to SMT-based model-checking algorithms.
  - ⇒ *DSS* is domain-independent.

Conclusion

- *DSS* can decompose a verification task into independent smaller tasks.
- *DSS* is domain-independent.
- *DSS* effectively distributes the workload to multiple processing units.

Supplementary webpage

References I

[1] Alt, L., Asadi, S., Chockler, H., Even-Mendoza, K., Fedyukovich, G., Hyvärinen, A.E.J., Sharygina, N.: HiFrog: SMT-based function summarization for software verification. In: Proc. TACAS. pp. 207–213. LNCS 10206 (2017). https://doi.org/10.1007/978-3-662-54580-5_12

[2] Beyer, D.: State of the art in software verification and witness validation: SV-COMP 2024. In: Proc. TACAS (3). pp. 299–329. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_15

[3] Beyer, D., Cimatti, A., Griggio, A., Keremoglu, M.E., Sebastiani, R.: Software model checking via large-block encoding. In: Proc. FMCAD. pp. 25–32. IEEE (2009). https://doi.org/10.1109/FMCAD.2009.5351147

[4] Beyer, D., Friedberger, K.: Domain-independent interprocedural program analysis using block-abstraction memoization. In: Proc. ESEC/FSE. pp. 50–62. ACM (2020). https://doi.org/10.1145/3368089.3409718

[5] Beyer, D., Kettl, M., Lemberger, T.: Decomposing software verification using distributed summary synthesis. Proc. ACM Softw. Eng. **1**(FSE) (2024). https://doi.org/10.1145/3660766

[6] Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., O'Hearn, P.W., Papakonstantinou, I., Purbrick, J., Rodriguez, D.: Moving fast with software verification. In: Proc. NFM. pp. 3–11. LNCS 9058, Springer (2015). https://doi.org/10.1007/978-3-319-17524-9_1

References II

[7] Kettl, M., Lemberger, T.: The static analyzer Infer in SV-COMP (competition contribution). In: Proc. TACAS (2). pp. 451–456. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_30