Decomposing Software Verification using Distributed Summary Synthesis

Dirk Beyer, **Matthias Kettl**, Thomas Lemberger

2024-07-19

LMU Munich, Germany

Motivation

Formal verification is a time-consuming task.

Long response times hinder easy integration.

No silver bullet has been found yet.

Motivation

Formal verification is a time-consuming task.

Long response times hinder easy integration.

No silver bullet has been found yet.

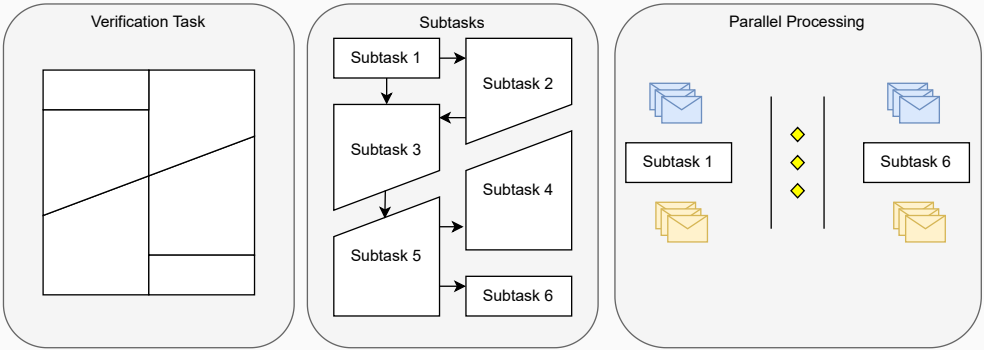
⇒ Distributed summary synthesis (*DSS*) aims at scaling formal verification independent of the abstract domain.

Motivation

Existing approaches have limitations that distributed summary synthesis solves:

- INFER [4, 6] scales well but reports many false alarms.
  ⇒ *DSS* inherits all properties of the underlying analysis.
- BAM [2] 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.

Overview

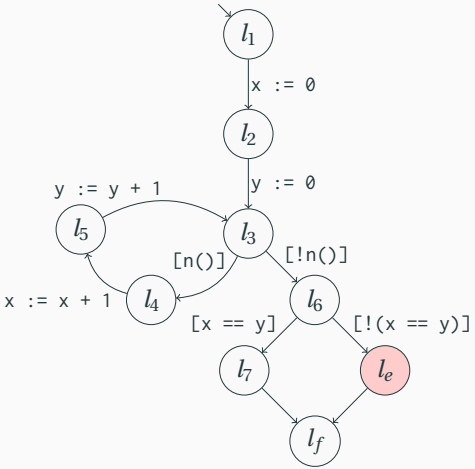


Overview of the *DSS* approach

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

Decompositions

We split a large verification task into multiple subtasks.

Requirements for a good decomposition:

- Each block has exactly one entry and one exit location.
- Loops should be reflected as loops in the block graph.
- Blocks should be as large as possible.

Decompositions

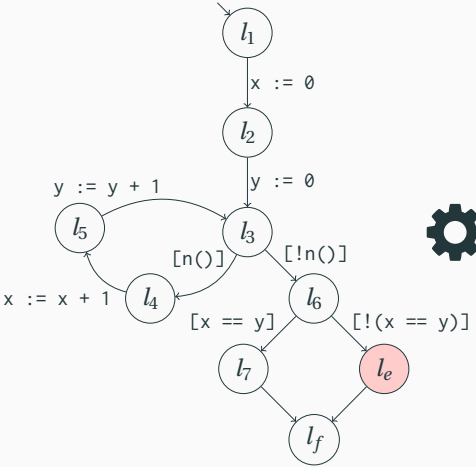
We split a large verification task into multiple subtasks.

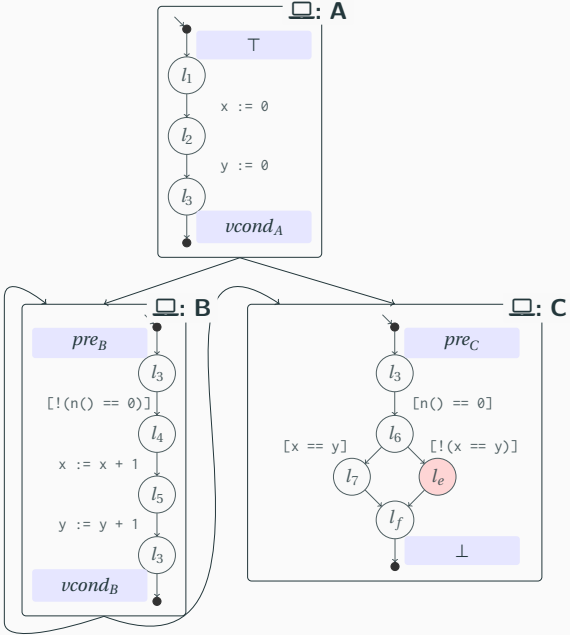
Requirements for a good decomposition:

- Each block has exactly one entry and one exit location.
- Loops should be reflected as loops in the block graph.
- Blocks should be as large as possible.

Approach: We decompose the CFA at every join/split location and eagerly merge horizontally and vertically.

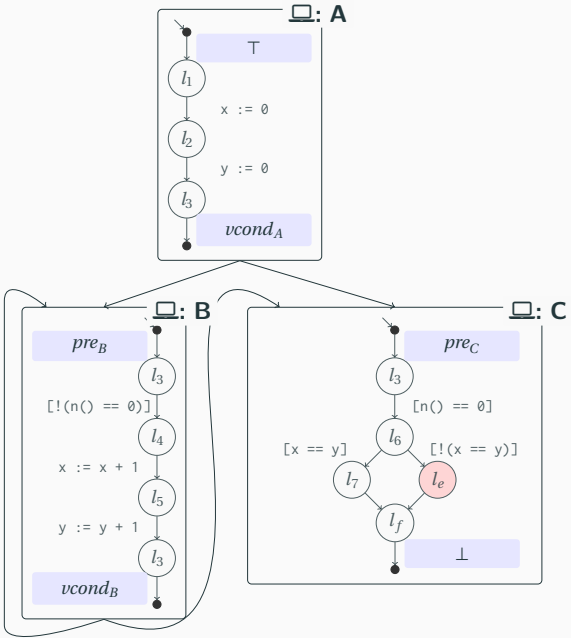
Decompositions





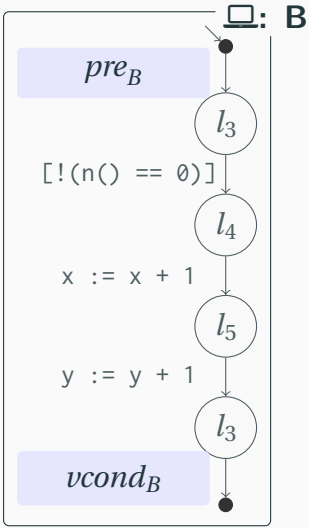
Communication Model

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



Workers

- Preconditions summarize the state space of a block (eventually over-approximating).
- All violation conditions need to be refuted to prove a program safe.
- Preconditions are refined until all violation conditions are refuted or at least one is confirmed.

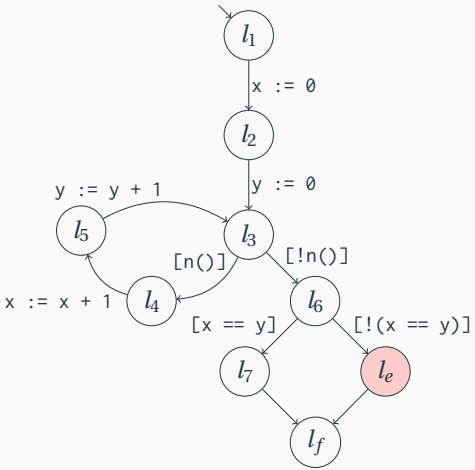


Traditional Predicate Analysis

Predicate analysis with LBE [3] and CEGAR [5] starts with an empty precision ($\pi = \{\}$) for l_3.

$$R = \{(l_1, (\top, \top)),$$

$$\}$$



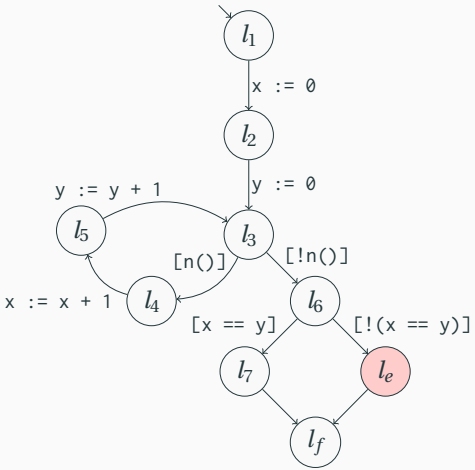
CFA of program

Traditional Predicate Analysis

Predicate analysis with LBE [3] and CEGAR [5] starts with an empty precision ($\pi = \{\}$) for l_3.

$$R = \{(l_1, (\top, \top)),$$
$$(l_2, (x = 0, \top)),$$

$$\}$$



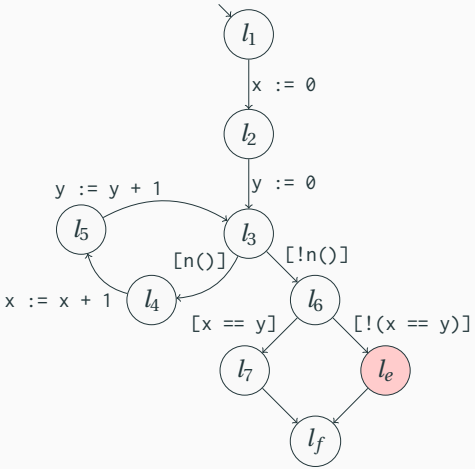
CFA of program

Traditional Predicate Analysis

Predicate analysis with LBE [3] and CEGAR [5] starts with an empty precision ($\pi = \{\}$) for l_3.

$$R = \{(l_1, (\top, \top)),$$
$$(l_2, (x = 0, \top)),$$
$$(l_3, (\top, \top)),$$

$$\}$$

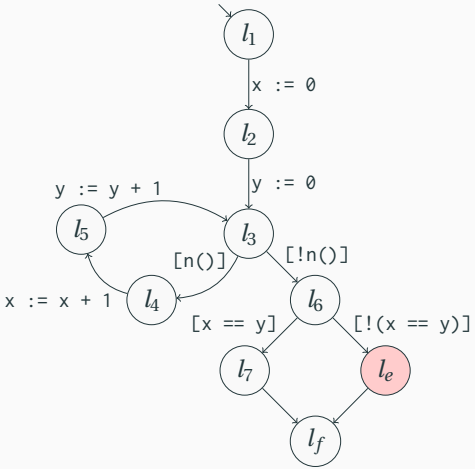


CFA of program

Traditional Predicate Analysis

Predicate analysis with LBE [3] and CEGAR [5] starts with an empty precision ($\pi = \{\}$) for l_3.

$$
\begin{aligned}
R = \{&(l_1, (\top, \top)),\\
&(l_2, (x = 0, \top)),\\
&(l_3, (\top, \top)),\\
&(l_4, (n(), \top)),\\
&\\
&\\
&\\
&\}
\end{aligned}
$$

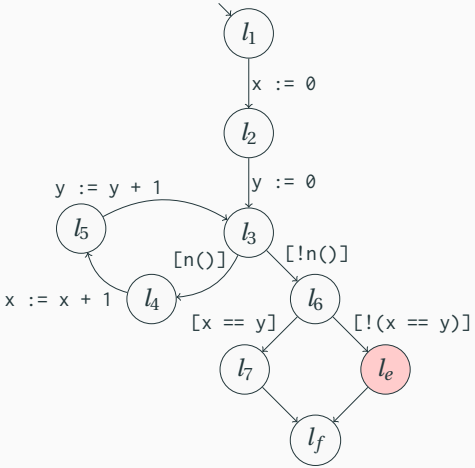


CFA of program

Traditional Predicate Analysis

Predicate analysis with LBE [3] and CEGAR [5] starts with an empty precision ($\pi = \{\}$) for l_3.

$$
\begin{aligned}
R = \{&(l_1, (\top, \top)),\\
&(l_2, (x = 0, \top)),\\
&(l_3, (\top, \top)),\\
&(l_4, (n(), \top)),\\
&(l_6, (\neg n(), \top)),\\
&\\
&\}
\end{aligned}
$$

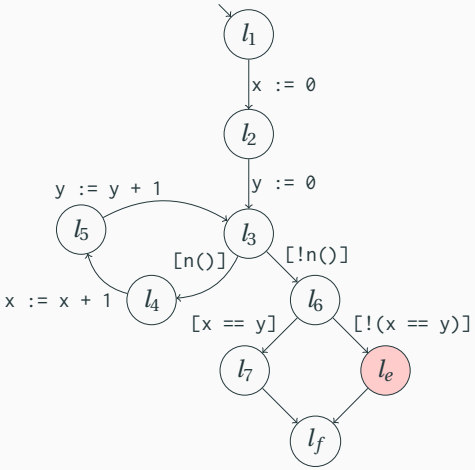


CFA of program

Traditional Predicate Analysis

Predicate analysis with LBE [3] and CEGAR [5] starts with an empty precision ($\pi = \{\}$) for l_3.

$$
\begin{aligned}
R = \{&(l_1, (\top, \top)),\\
&(l_2, (x = 0, \top)),\\
&(l_3, (\top, \top)),\\
&(l_4, (n(), \top)),\\
&(l_6, (\neg n(), \top)),\\
&(l_5, (n() \wedge x_1 = x_0 + 1, \top)),\\
&\}
\end{aligned}
$$

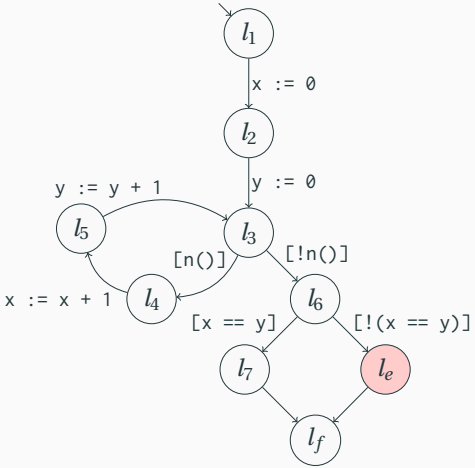


CFA of program

Traditional Predicate Analysis

Predicate analysis with LBE [3] and CEGAR [5] starts with an empty precision ($\pi = \{\}$) for l_3.

$$
\begin{aligned}
R = \{&(l_1, (\top, \top)),\\
&(l_2, (x = 0, \top)),\\
&(l_3, (\top, \top)),\\
&(l_4, (n(), \top)),\\
&(l_6, (\neg n(), \top)),\\
&(l_5, (n() \wedge x_1 = x_0 + 1, \top)),\\
&(l_e, (\neg n() \wedge x \neq y, \top))\}
\end{aligned}
$$



CFA of program

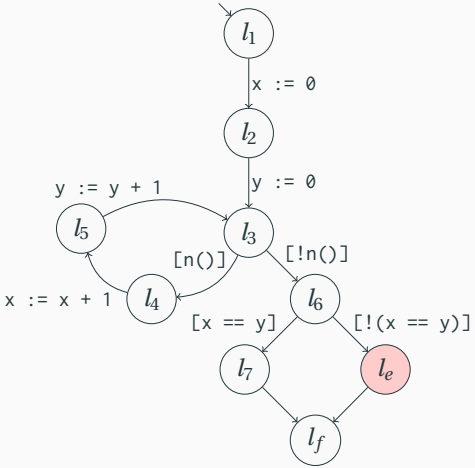
Traditional Predicate Analysis

Reconstruction of counterexample:

$$x = 0 \wedge y = 0 \wedge \neg n() \wedge x \neq y$$

$x = y$

Precision updated to $\pi = \{(x = y)\}$.



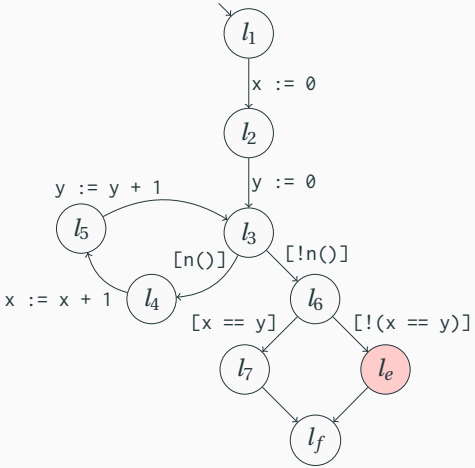
CFA of program

Traditional Predicate Analysis

Restart with precision $\pi = \{x = y\}$

$$R = \{(l_1, (\top, \top)),$$

$$\}$$



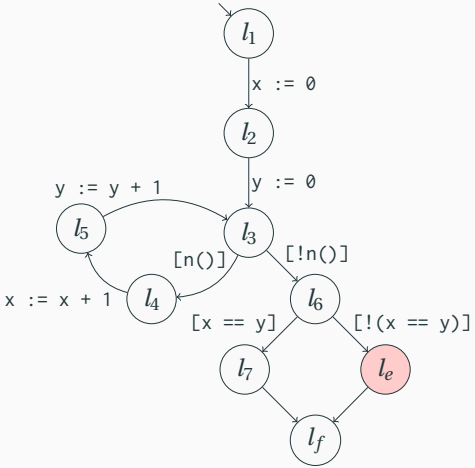
CFA of program

Traditional Predicate Analysis

Restart with precision $\pi = \{x = y\}$

$$R = \{(l_1, (\top, \top)),$$
$$(l_2, (x = 0, \top)),$$

$$\}$$

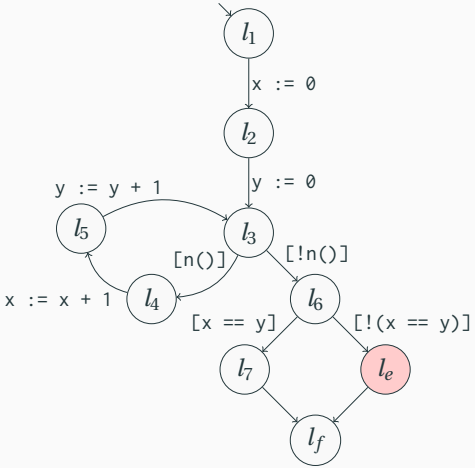


CFA of program

Traditional Predicate Analysis

Restart with precision $\pi = \{x = y\}$

$$R = \{(l_1, (\top, \top)),$$
$$(l_2, (x = 0, \top)),$$
$$(l_3, (\top, x = y)),$$
$$\}$$



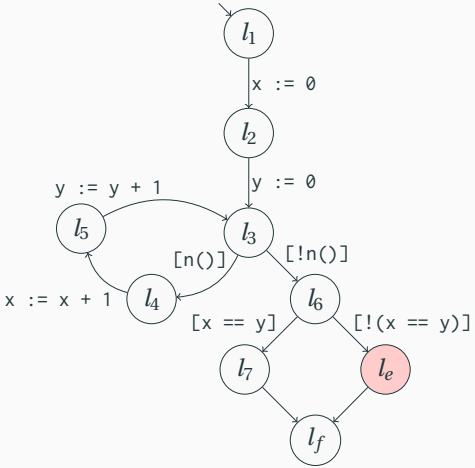
CFA of program

Traditional Predicate Analysis

Restart with precision $\pi = \{x = y\}$

$$R = \{(l_1, (\top, \top)),$$
$$(l_2, (x = 0, \top)),$$
$$(l_3, (\top, x = y)),$$
$$(l_4, (n(), x = y)),$$

$$\}$$

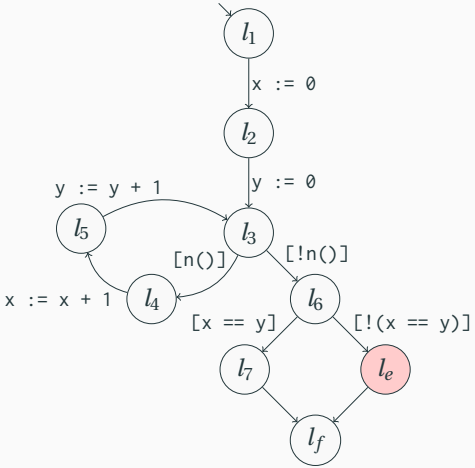


CFA of program

Traditional Predicate Analysis

Restart with precision $\pi = \{x = y\}$

$$
\begin{aligned}
R = \{&(l_1, (\top, \top)),\\
&(l_2, (x = 0, \top)),\\
&(l_3, (\top, x = y)),\\
&(l_4, (n(), x = y)),\\
&(l_6, (\neg n(), x = y)),\\
&\\
&\\
&\\
&\}
\end{aligned}
$$

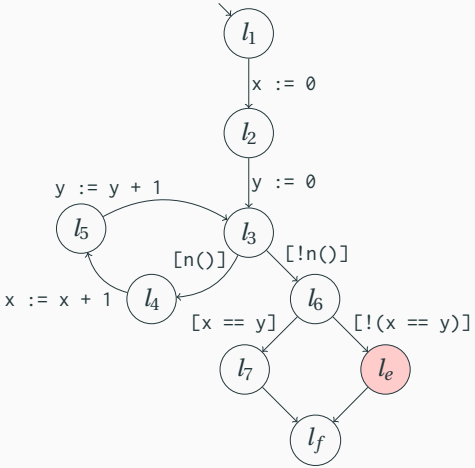


CFA of program

Traditional Predicate Analysis

Restart with precision $\pi = \{x = y\}$

$$
\begin{aligned}
R = \{&(l_1, (\top, \top)),\\
&(l_2, (x = 0, \top)),\\
&(l_3, (\top, x = y)),\\
&(l_4, (n(), x = y)),\\
&(l_6, (\neg n(), x = y)),\\
&(l_5, (n() \wedge x_1 = x_0 + 1, x = y)),\\
&\\
&\}
\end{aligned}
$$

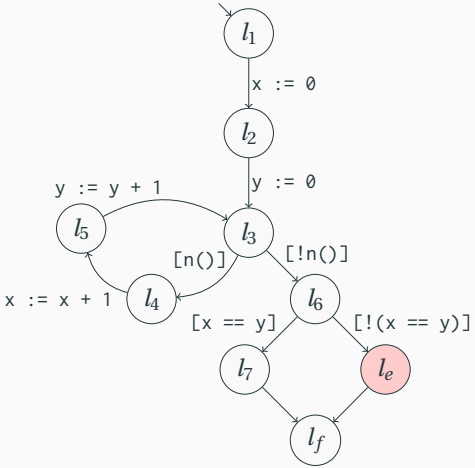


CFA of program

Traditional Predicate Analysis

Restart with precision $\pi = \{x = y\}$

$$
\begin{aligned}
R = \{&(l_1, (\top, \top)),\\
&(l_2, (x = 0, \top)),\\
&(l_3, (\top, x = y)),\\
&(l_4, (n(), x = y)),\\
&(l_6, (\neg n(), x = y)),\\
&(l_5, (n() \wedge x_1 = x_0 + 1, x = y)),\\
&(l_7, (\neg n() \wedge x = y, x = y)),\\
&\\
&\}
\end{aligned}
$$

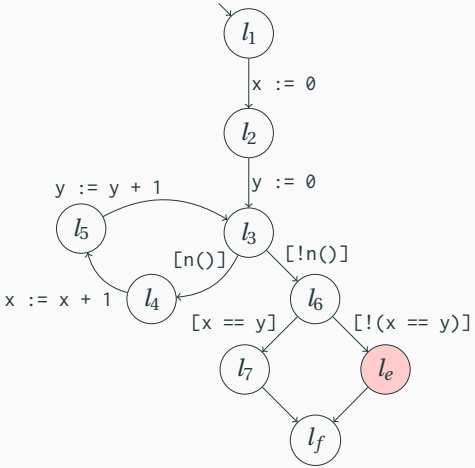


CFA of program

Traditional Predicate Analysis

Restart with precision $\pi = \{x = y\}$

$$
\begin{aligned}
R = \{&(l_1, (\top, \top)),\\
&(l_2, (x = 0, \top)),\\
&(l_3, (\top, x = y)),\\
&(l_4, (n(), x = y)),\\
&(l_6, (\neg n(), x = y)),\\
&(l_5, (n() \wedge x_1 = x_0 + 1, x = y)),\\
&(l_7, (\neg n() \wedge x = y, x = y)),\\
&(l_3, (\top, x = y)),\\
&\}
\end{aligned}
$$

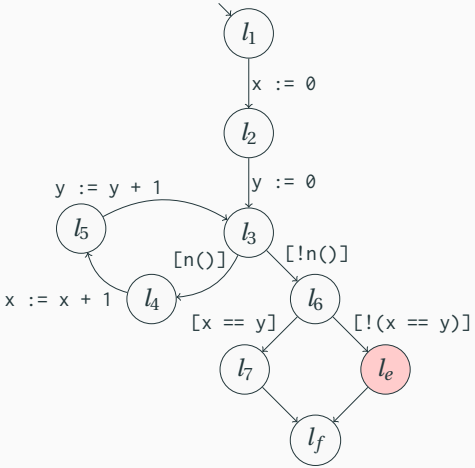


CFA of program

Traditional Predicate Analysis

Restart with precision $\pi = \{x = y\}$

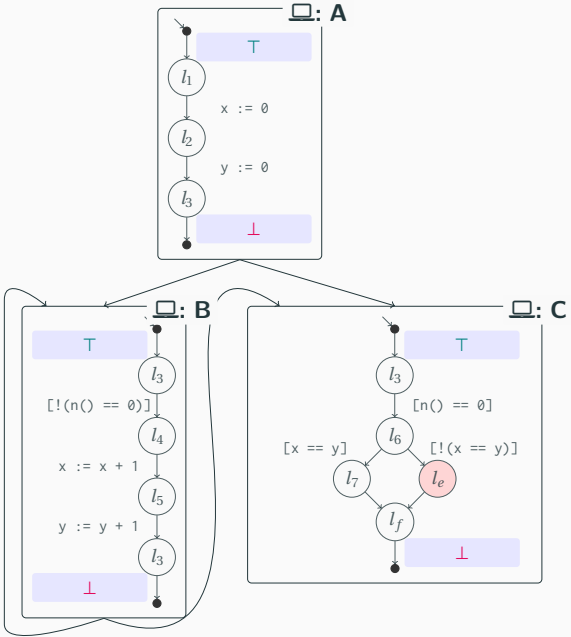
$$
\begin{aligned}
R = \{&(l_1, (\top, \top)),\\
&(l_2, (x = 0, \top)),\\
&(l_3, (\top, x = y)),\\
&(l_4, (n(), x = y)),\\
&(l_6, (\neg n(), x = y)),\\
&(l_5, (n() \wedge x_1 = x_0 + 1, x = y)),\\
&(l_7, (\neg n() \wedge x = y, x = y)),\\
&(l_3, (\top, x = y)),\\
&(l_f, (\neg n() \wedge x = y, x = y))\}
\end{aligned}
$$



CFA of program

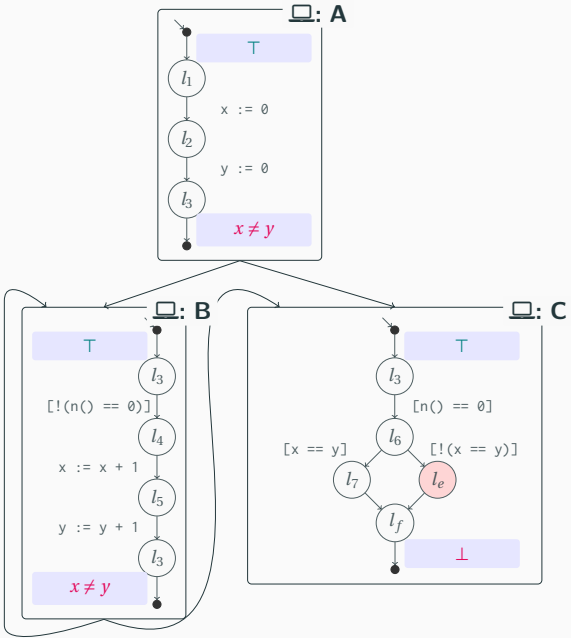
Verification with *DSS*

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



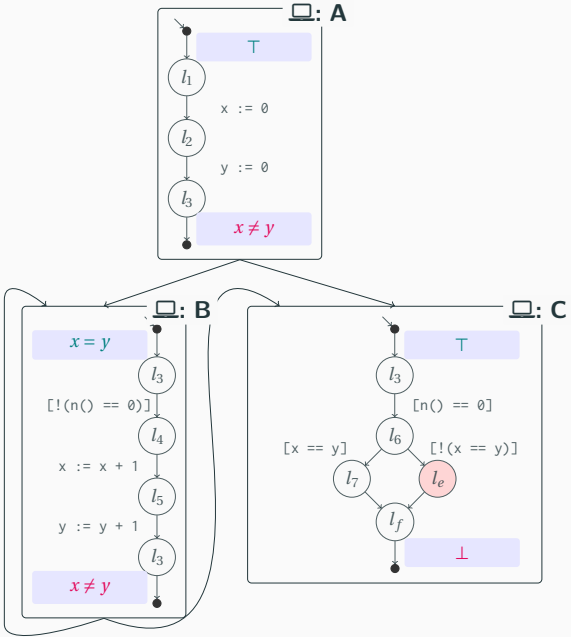
Verification with *DSS*

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



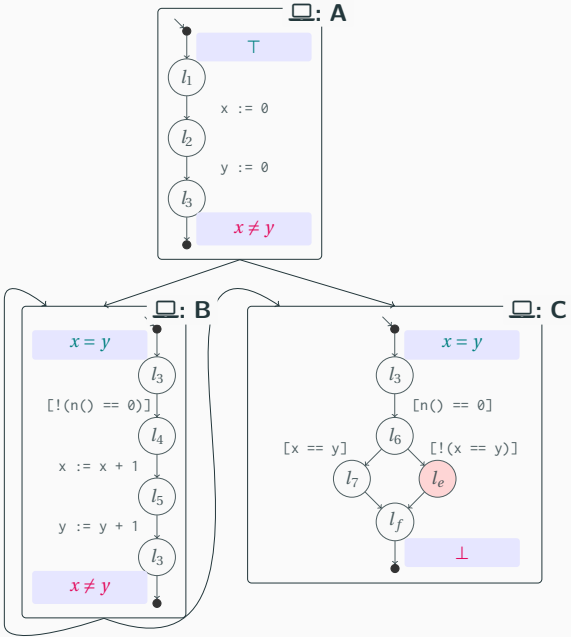
Verification with *DSS*

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



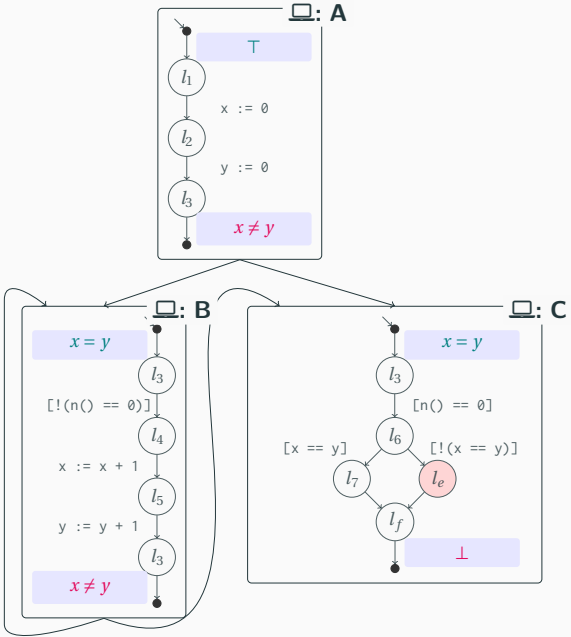
Verification with *DSS*

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



Verification with *DSS*

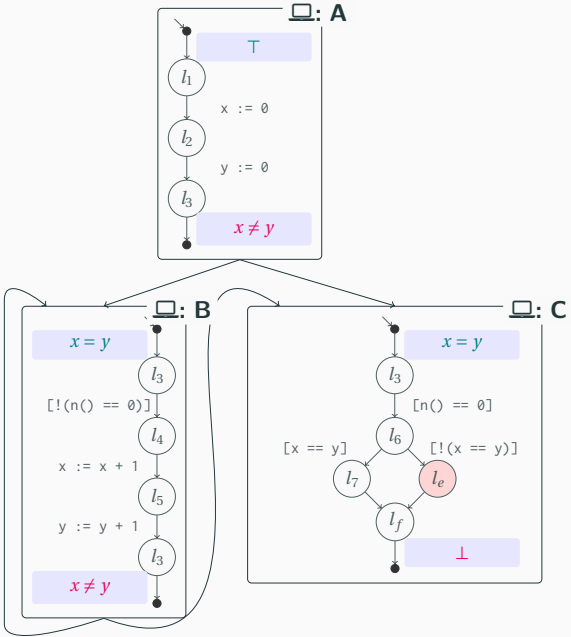
Block	Result
A	*idle*
B	*idle*
C	*idle*



Verification with *DSS*

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

⇒ Fixed point reached, program safe

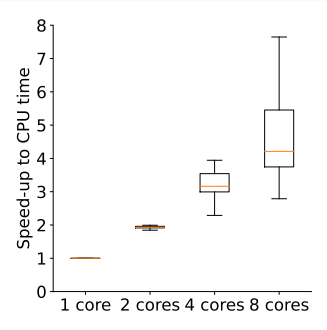


Evaluation: Setup

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 benchmark setup: 15 GB RAM and an 8 core Intel Xeon E3-1230 v5 with 3.40 GHz.

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	$\text{CPU}_{\mathbb{P}}(s)$	$\text{CPU}_{DSS}(s)$	$\text{RT}_{\mathbb{P}}(s)$	$\mathbf{RT}_{DSS}(s)$	# 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.

Conclusion

- *DSS* is a parallel domain-independent software verification approach.
- *DSS* consumes more memory and has to invest time to (un)pack messages.
- *DSS* effectively distributes the workload to multiple processing units and outperforms predicate analysis on some tasks.

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., 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

[3] Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate abstraction with adjustable-block encoding. In: Proc. FMCAD. pp. 189–197. FMCAD (2010)

References ii

[4] 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

[5] Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Proc. CAV. pp. 154–169. LNCS 1855, Springer (2000). https://doi.org/10.1007/10722167_15

[6] 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