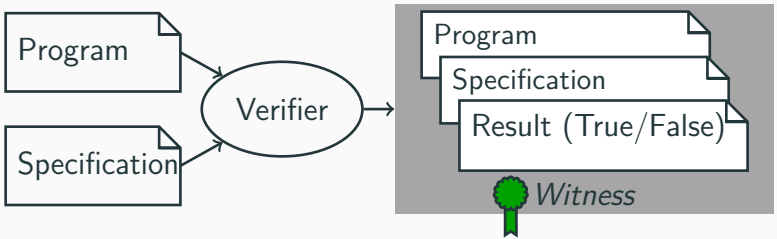
Distributed Summary Synthesis: An Approach for Computing Block Contracts in Software Model Checking

Dirk Beyer

LMU Munich, Germany

ftsrg@BMI-MIT, Budapest
2025-09-17

Automatic Software Verification



Motivation

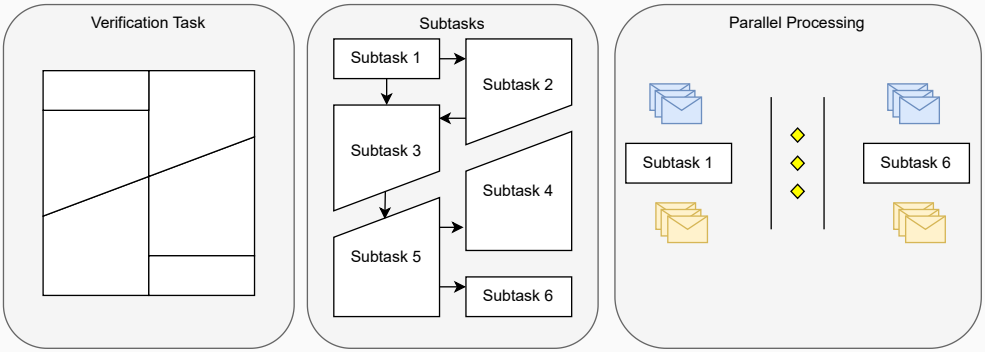
- We need low response time.
- Our solution: Massively parallel approaches.
- Solution: Decomposition into blocks, construct block-wise contracts through backend verifier
- Goal: Scalable and Distributed Software Verification

Based on [6]:

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 the *DSS* approach

Decomposition

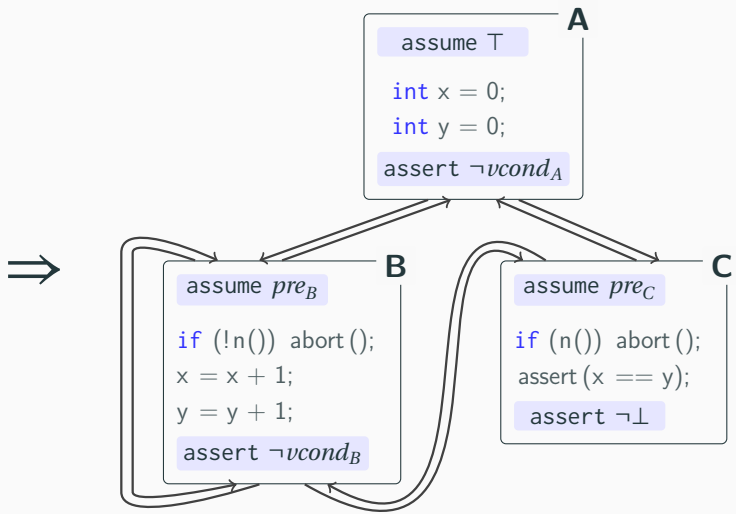
Requirements for eligible decompositions:

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

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

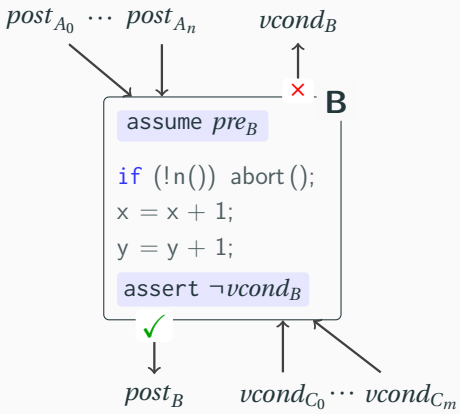
Decomposition

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



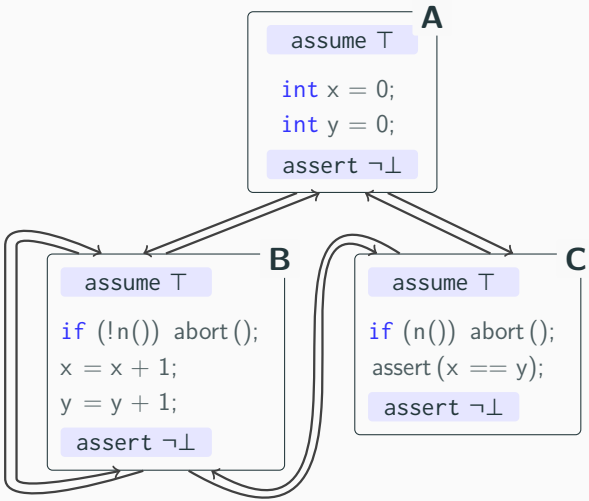
Workers

- 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.



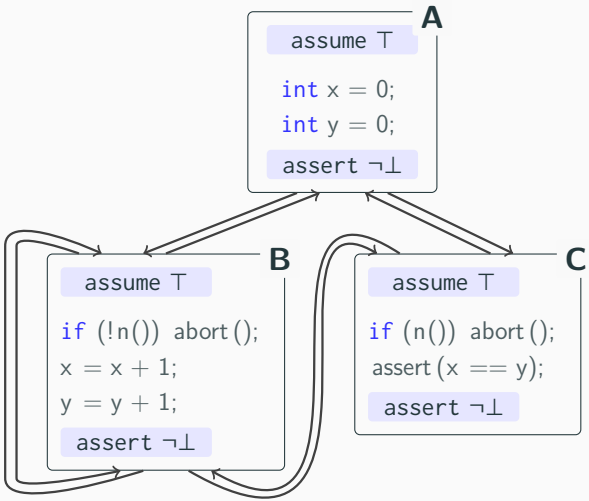
DSS Example

it	**Block**	*pre*	*vcond*	**Result**



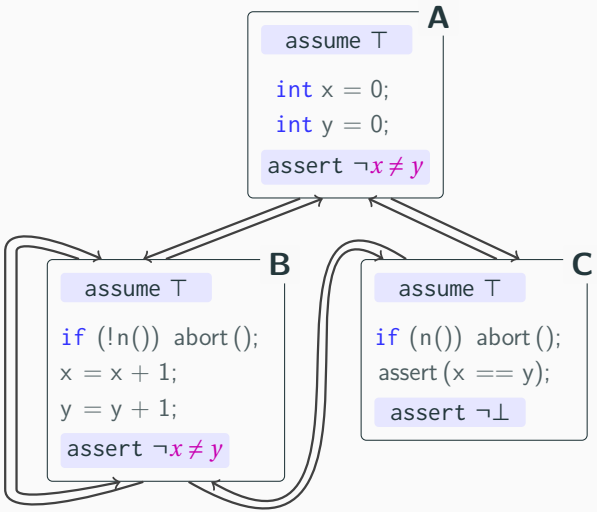
DSS Example

it	Block	*pre*	*vcond*	Result
1	A	$\top$	$\bot$	$\downarrow\boxtimes: \top$
1	B	$\top$	$\bot$	$\downarrow\boxtimes: \top$
1	C	$\bot$	$\top$	$\uparrow\boxtimes: x \neq y$



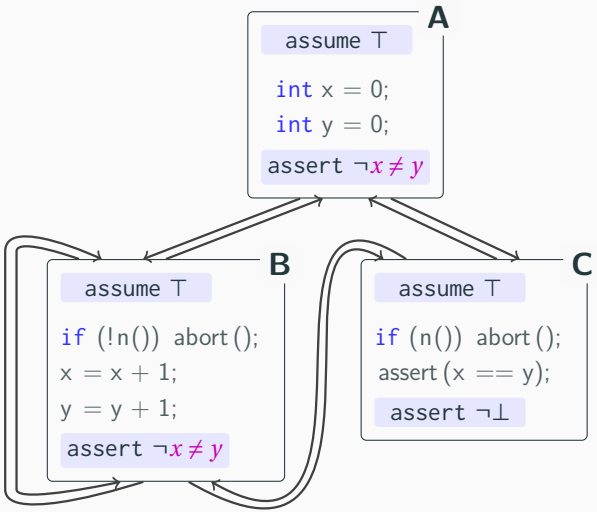
DSS Example

it	Block	*pre*	*vcond*	Result
1	A	$\top$	$\bot$	$\downarrow\boxtimes: \top$
1	B	$\top$	$\bot$	$\downarrow\boxtimes: \top$
1	C	$\bot$	$\top$	$\uparrow\boxtimes: x \neq y$



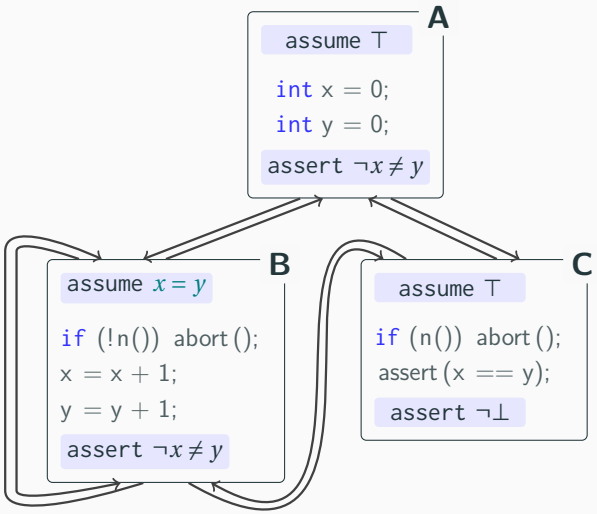
DSS Example

it	**Block**	*pre*	*vcond*	**Result**
1	A	$\top$	$\bot$	$\downarrow\boxtimes$: $\top$
1	B	$\top$	$\bot$	$\downarrow\boxtimes$: $\top$
1	C	$\bot$	$\top$	$\uparrow\boxtimes$: $x \neq y$
2	A	$\top$	$x \neq y$	$\downarrow\boxtimes$: $x = y$
2	B	$\top$	$x \neq y$	$\uparrow\boxtimes$: $x \neq y$



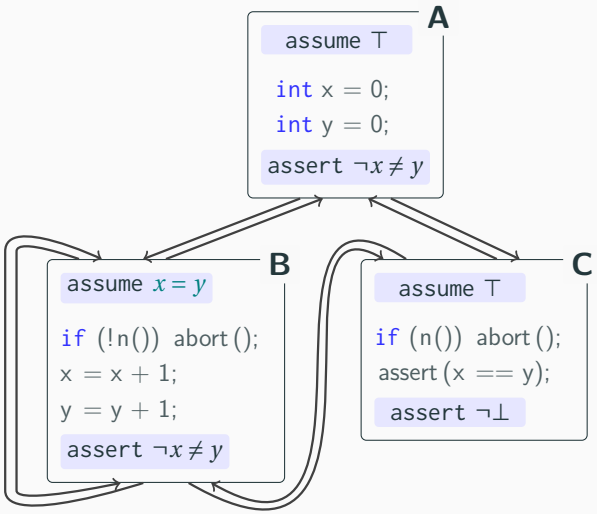
DSS Example

it	Block	*pre*	*vcond*	Result
1	A	$\top$	$\bot$	$\downarrow\boxtimes$: $\top$
1	B	$\top$	$\bot$	$\downarrow\boxtimes$: $\top$
1	C	$\bot$	$\top$	$\uparrow\boxtimes$: $x \neq y$
2	A	$\top$	$x \neq y$	$\downarrow\boxtimes$: $x = y$
2	B	$\top$	$x \neq y$	$\uparrow\boxtimes$: $x \neq y$



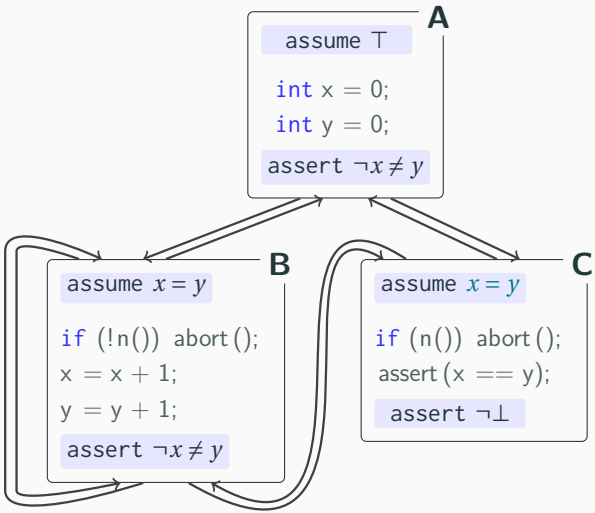
DSS Example

it	Block	*pre*	*vcond*	Result
1	A	$\top$	$\bot$	$\downarrow\boxtimes$: $\top$
1	B	$\top$	$\bot$	$\downarrow\boxtimes$: $\top$
1	C	$\bot$	$\top$	$\uparrow\boxtimes$: $x \neq y$
2	A	$\top$	$x \neq y$	$\downarrow\boxtimes$: $x = y$
2	B	$\top$	$x \neq y$	$\uparrow\boxtimes$: $x \neq y$
3	B	$x = y$	$x \neq y$	$\downarrow\boxtimes$: $x = y$



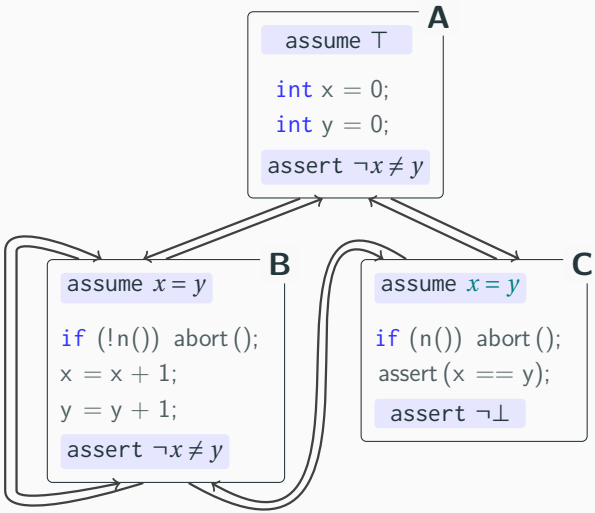
DSS Example

it	Block	*pre*	*vcond*	Result
1	A	$\top$	$\bot$	$\downarrow\boxtimes$: $\top$
1	B	$\top$	$\bot$	$\downarrow\boxtimes$: $\top$
1	C	$\bot$	$\top$	$\uparrow\boxtimes$: $x \neq y$
2	A	$\top$	$x \neq y$	$\downarrow\boxtimes$: $x = y$
2	B	$\top$	$x \neq y$	$\uparrow\boxtimes$: $x \neq y$
3	B	$x = y$	$x \neq y$	$\downarrow\boxtimes$: $x = y$



DSS Example

it	Block	*pre*	*vcond*	Result
1	A	$\top$	$\bot$	$\downarrow\boxtimes$: $\top$
1	B	$\top$	$\bot$	$\downarrow\boxtimes$: $\top$
1	C	$\bot$	$\top$	$\uparrow\boxtimes$: $x \neq y$
2	A	$\top$	$x \neq y$	$\downarrow\boxtimes$: $x = y$
2	B	$\top$	$x \neq y$	$\uparrow\boxtimes$: $x \neq y$
3	B	$x = y$	$x \neq y$	$\downarrow\boxtimes$: $x = y$
4	C	$x = y$	$\bot$	$\downarrow\boxtimes$: $\top$



Details on Information Exchange

- Subtasks identified through precise program location
- Message consists of
  relevant program location + (postcondition | violation condition)
- Postcondition:
  - Craig interpolant at block end
  - Communicated in SMT-LIB
- Violation conditions:
  - Precise path formula
  - Communicated in SMT-LIB

Related Approaches

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

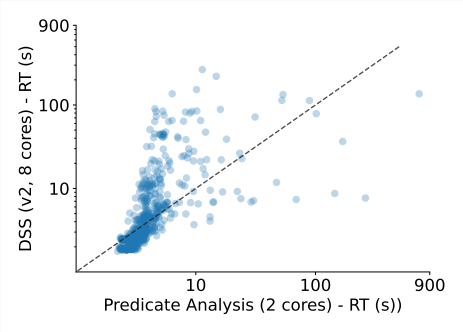
- INFER [7, 8] scales well but reports many false alarms.
  ⇒ *DSS* is not restricted to a local context.
- BAM [4] has nested blocks that are not parallelizable.
  ⇒ *DSS* avoids nested blocks and parallelizes as much as possible.
- HIFROG [1] is bound to function summaries and SMT-based algorithms.
  ⇒ *DSS* is domain-independent and can use arbitrary block size.

Evaluation: Setup

Benchmark Setup:

- We implemented DSS in CPAchecker [5].
- 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.

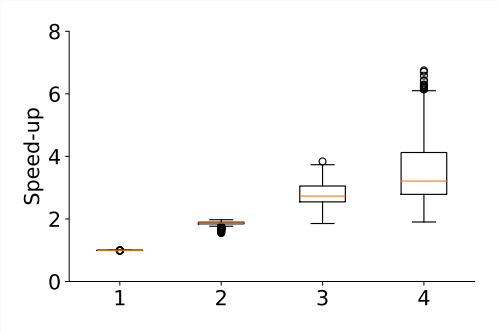
Evaluation: Results



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	$\text{CPU}_{\mathbb{P}}(s)$	$\text{CPU}_{DSS}(s)$	$\text{RT}_{\mathbb{P}}(s)$	$\mathbf{RT}_{DSS}(s)$	Block size (avg.)	# Threads
input–touchscreen–...	200	200	170	37	16	113
mtd–devices–slram....	130	440	100	79	16	94
power–wm831x_backu...	780	840	730	140	14	62
regulator–wm831x-ld...	68	26	47	12	9.7	353
rtc–rtc-ds1553.ko-l...	49	19	30	7.1	15	163

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

Conclusion

- *DSS* can decompose a verification task into independent smaller tasks.
- *DSS* is domain-independent and supports various block sizes.
- *DSS* effectively distributes the workload to multiple processing units.
- *DSS* is implemented in CPAchecker [5].

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., 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

[6] 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

References II

[7] 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

[8] 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