Distributed Automatic Contract Construction

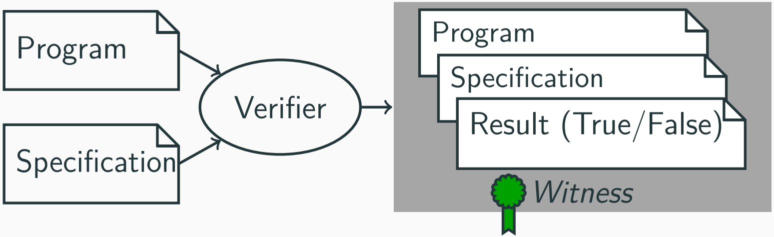
Dirk Beyer
LMU Munich, Germany

25 Years KeY
Bad Herrenalb
2024-07-31

Happy Birthday, KeY

- Part 1: Distributed Automatic Contract Construction
- Part 2: Find, Use, and Conserve KeY

Automatic Software Verification



Mostly context-sensitive, whole-program analysis

Motivation Part 1

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

Solution: Distributed Summary Synthesis

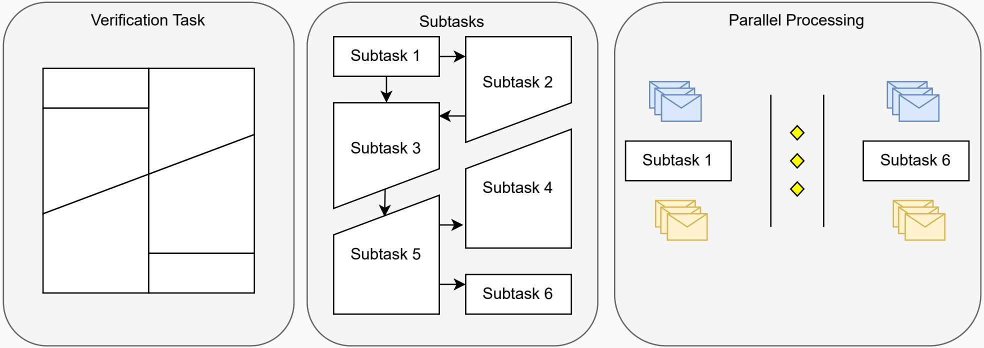
Based on [5]:

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

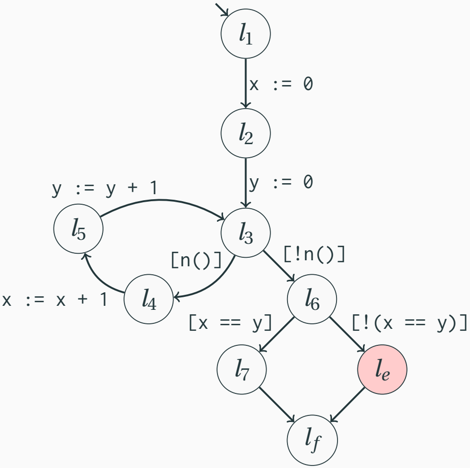


Overview of the *DSS* approach

Example: Control-Flow Automaton

```c
int main() {
  int x = 0;
  int y = 0;
  while (n()) {
    x++;
    y++;
  }
  assert(x == y);
}
```

Safe program



CFA of program

Decomposition

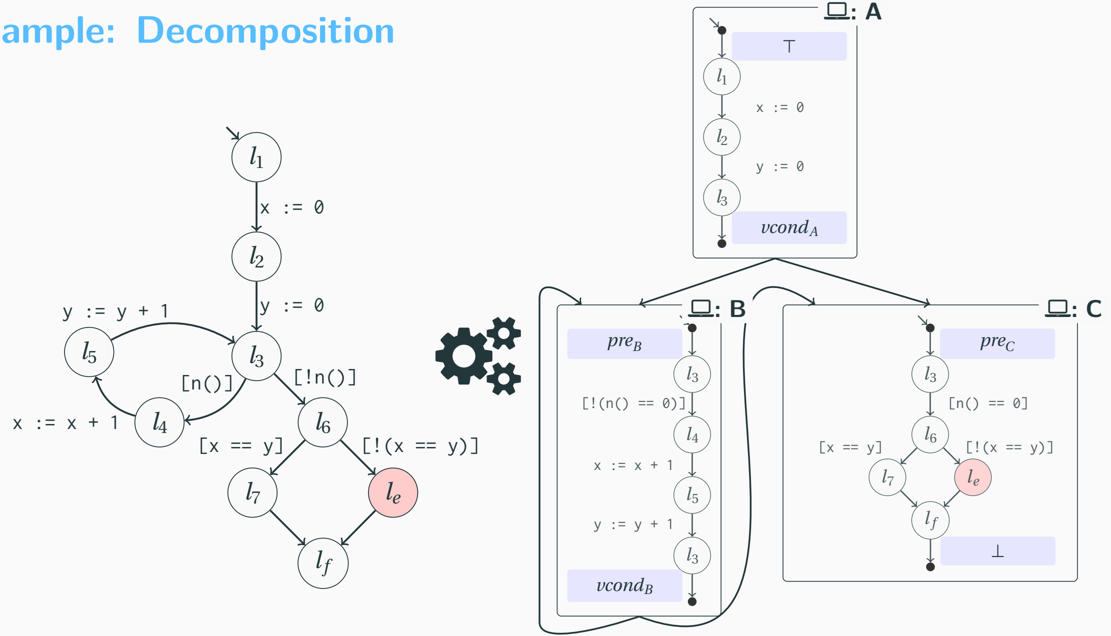
We split a large verification task into multiple smaller subtasks.

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.

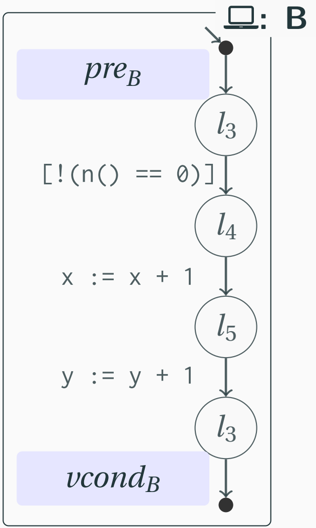
Approach: We decompose the CFA similar to large-block encoding [3].

Example: Decomposition



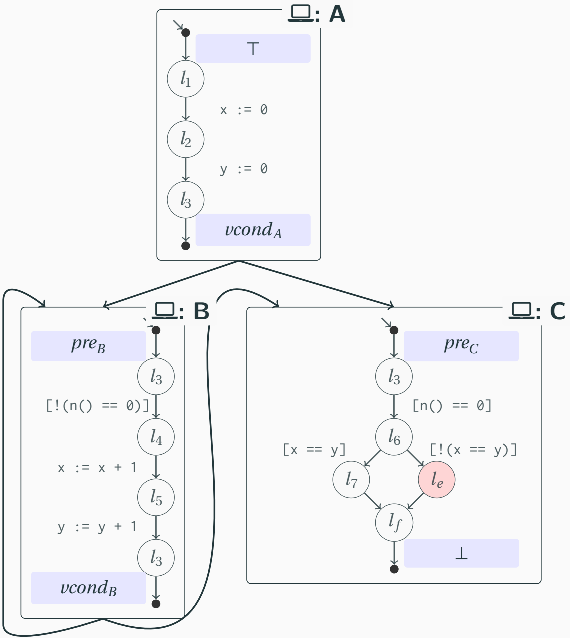
Workers

- 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.



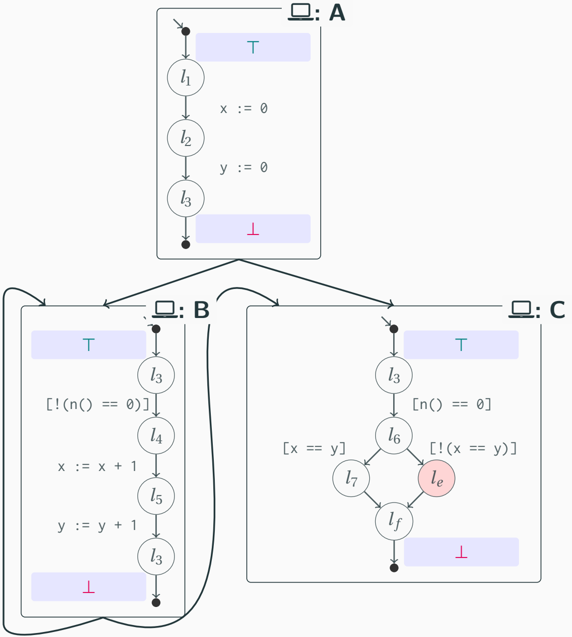
Communication Model

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



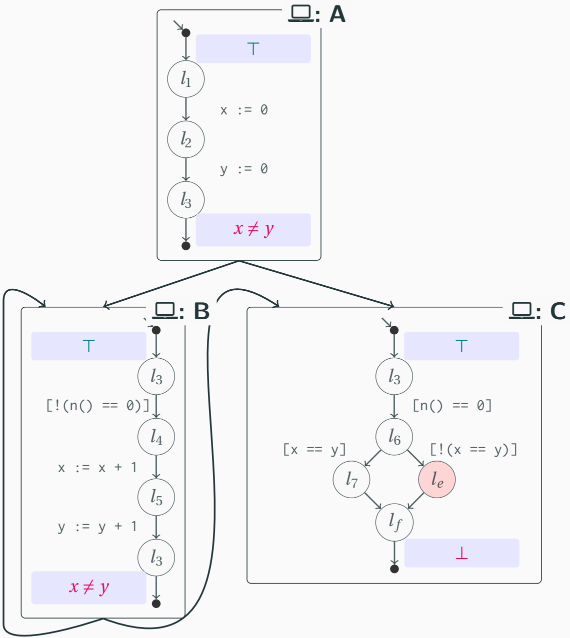
Verification with *DSS* 1

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



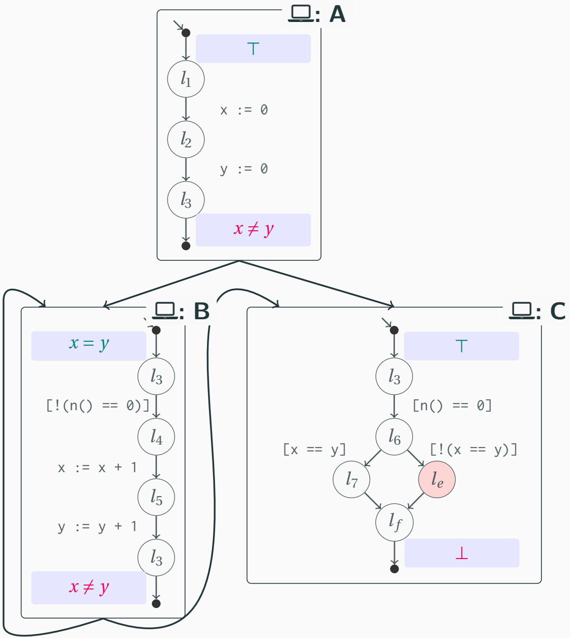
Verification with *DSS* 2

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



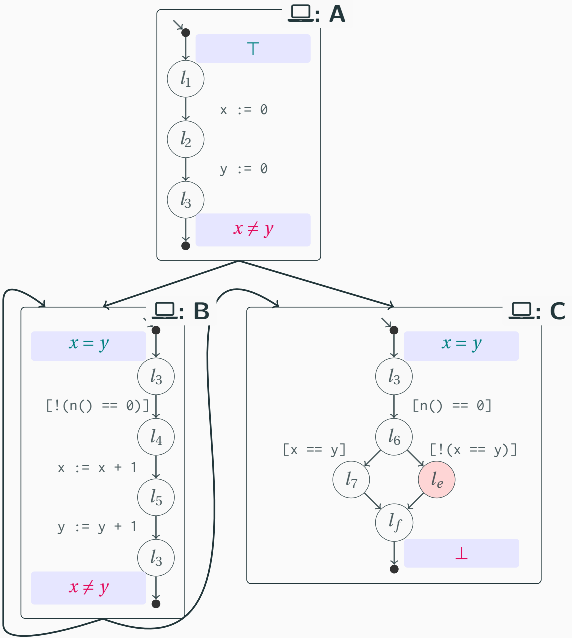
Verification with *DSS* 3

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



Verification with *DSS* 4

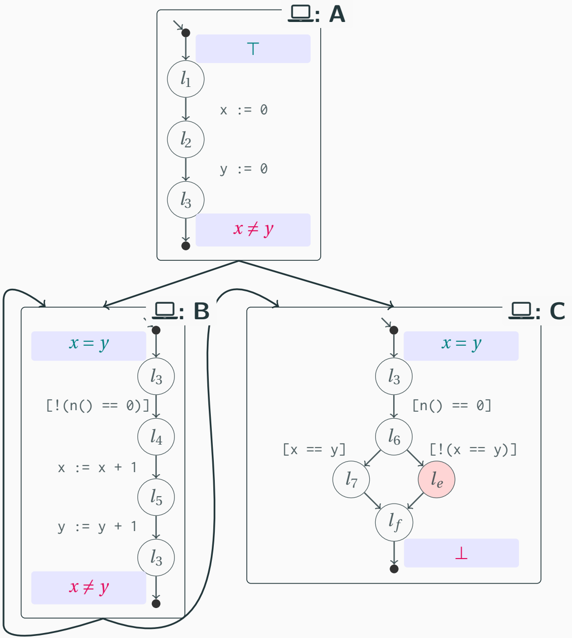
Block	Result
A	*idle*
B	*idle*
C	↓✉$_\emptyset$: $\top$



Verification with *DSS* 5

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

⇒ Fix-point reached, program safe.

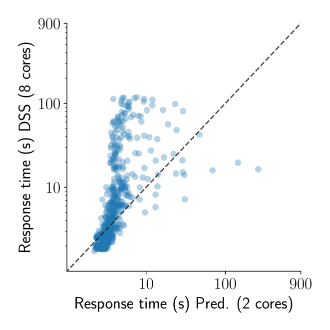


Evaluation: Setup

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.

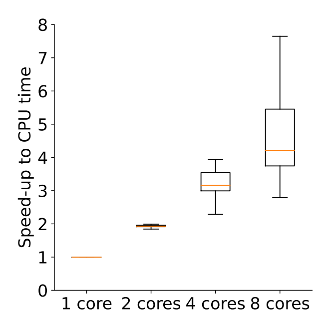
Evaluation: Results



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.

Conclusion of Part 1

- *DSS* is a domain-independent software-verification approach.
- *DSS* effectively distributes the workload to multiple processing units.

Supplementary webpage

Part 2

- **Conserve KeY**

Motivation Part 2

- **Find**: Which tools for software verification exist?
- ... for test-case generation?
- ... for SMT solving?
- ... for hardware verification?
- **Reuse**: How to get executables?
- Where to find documentation?
- Am I allowed to use it?
- How to use them?
- **Conserve**: Which operating system, libraries, environment?

Requirements for Solution

- Support documentation and reuse
- Easy to query and generate knowledge base
- Long-term availability/executability of tools
- Must come with tool support
- Approach must be compatible with competitions

Solution [1]

One central repository:

https://gitlab.com/sosy-lab/benchmarking/fm-tools

which gives information about:

- Location of the tool (via DOI, just like other literature)
- License
- Contact (via ORCID)
- Project web site
- Options
- Requirements (certain Docker container / VM)

Maintained by formal-methods community

Example: Entry for KeY

```yaml
name: KeY
input_languages:
  - Java
project_url: https://www.key-project.org/
repository_url: https://github.com/KeYProject/key
spdx_license_identifier: GPL-2.0
benchexec_toolinfo_module: "benchexec.tools.key_cli"
fmtools_format_version: "2.0"
fmtools_entry_maintainers:
  - ricffb
```

Example: KeY's Contacts


```yaml
maintainers:
  - orcid: 0000-0002-5671-2555
    name: Wolfgang Ahrendt
    institution: Chalmers University of Technology
    country: Sweden
    url: https://www.cse.chalmers.se/~ahrendt/
  - orcid: 0000-0002-9672-3291
    name: Bernhard Beckert
    institution: Karlsruhe Institute of Technology
    country: Germany
    url: https://formal.kastel.kit.edu/beckert/
  - orcid: 0000-0001-8000-7613
    name: Reiner Hähnle
    institution: TU Darmstadt
    country: Germany
    url: https://www.informatik.tu-darmstadt.de/se/
      gruppenmitglieder/groupmembers_detailseite_30784.en.jsp
  - orcid: 0000-0002-2350-1831
    name: Mattias Ulbrich
    institution: Karlsruhe Institute of Technology
    country: Germany
    url: https://formal.kastel.kit.edu/ulbrich/
  - orcid: 0000-0001-8446-4598
```

Example: KeY's Versions


```yaml
versions:
  - version: "2.13"
    doi: 10.5281/zenodo.12945286
    benchexec_toolinfo_options: []
    required_ubuntu_packages:
      - openjdk-21-jre-headless
    base_container_images:
      - ubuntu:22.04
```

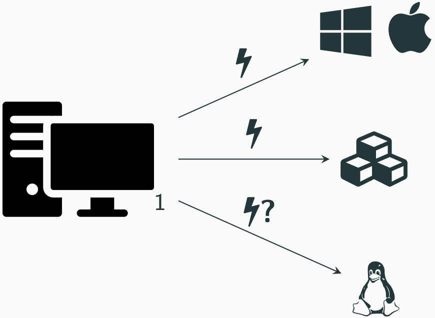
Example: KeY's Documentation


```yaml
literature:
  - doi: 10.1007/978-3-030-64354-6
    title: "Deductive␣Software␣Verification:␣Future␣Perspectives␣-
        ␣Reflections␣on␣the␣Occasion␣of␣20␣Years␣of␣KeY"
    year: 2020
  - doi: 10.1007/978-3-319-49812-6
    title: "Deductive␣Software␣Verification␣-␣The␣KeY␣Book␣-␣From␣
        Theory␣to␣Practice"
    year: 2016
  - doi: 10.1007/978-3-319-12154-3_4
    title: "The␣KeY␣Platform␣for␣Verification␣and␣Analysis␣of␣Java
        ␣Programs"
    year: 2014
  - doi: 10.1007/s10270-004-0058-x
    title: "The␣KeY␣Tool"
    year: 2005
  - doi: 10.1007/3-540-40006-0_3
    title: "The␣KeY␣Approach:␣Integrating␣Object␣Oriented␣Design␣
        and␣Formal␣Verification"
    year: 2000
```

FM-Tools is FAIR

- **F**indable:
  overview is available on internet,
  generated knowledge base
- **A**ccessible:
  data retrievable via Git, format is YAML
- **I**nteroperable:
  Format is defined in schema,
  archives identified by DOIs, researchers by ORCIDs
- **R**eusable:
  Data are CC-BY, each tool comes with a license,
  format of tool archive standardized

What about the Environment?



[1] Image: Flaticon.com

FM-Weck: Run Tools in Conserved Environment [6, Proc. FM 2024]

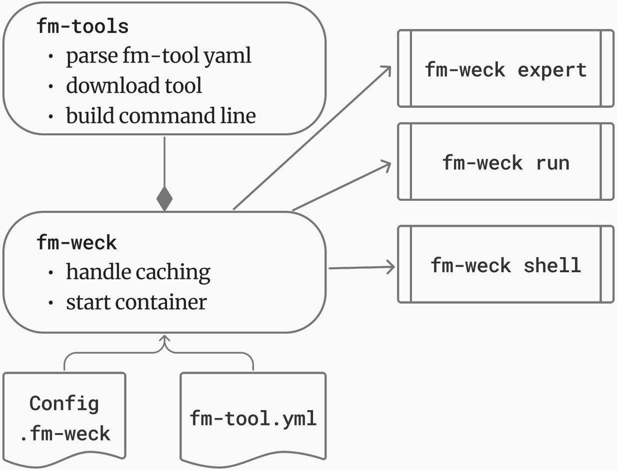
Refer to known fm-tools by identifier:version

`fm-weck` `run` `key:2.13` `BinarySearch`

Download, install, and run the tool

- No knowledge of the tools CLI needed
- Tool runs in a container (no dependencies on host system)

FM-Weck: Architecture



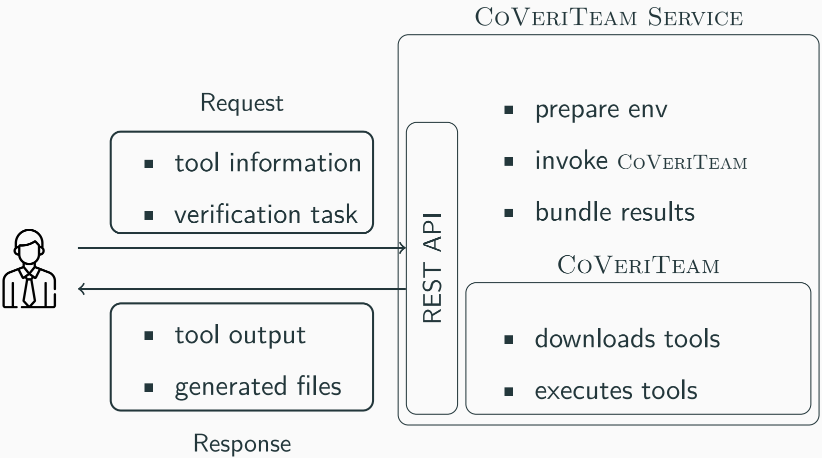
FM-Weck: Modes of Operation

- Download and execute tool in container
- No knowledge of tool needed

- Download and execute tool in container
- Expert knowledge about tool required

- Spin up interactive shell in tool environment

CoVeriTeam Service: Run Tool as Web Service [4, Proc. ICSE 2023, companion]



Conclusion

FM-Tools collects and stores essential information to:

- Run a tool as web service via CoVeriTeam Service [4]
- Run a tool in conserved environment via FM-Weck [6]
- Generate a knowledge base about formal-methods tools [1]
  https://fm-tools.sosy-lab.org

https://gitlab.com/sosy-lab/benchmarking/fm-tools

References I

[1] Beyer, D.: Conservation and accessibility of tools for formal methods. In: Proc. Festschrift Podelski 65th Birthday. Springer (2024)

[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., Kanav, S., Wachowitz, H.: CoVeriTeam Service: Verification as a service. In: Proc. ICSE, companion. pp. 21–25. IEEE (2023). https://doi.org/10.1109/ICSE-Companion58688.2023.00017

[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] Beyer, D., Wachowitz, H.: FM-Weck: Containerized execution of formal-methods tools. In: Proc. FM. LNCS, Springer (2024)