FM-Tools: A Library of Tools for Formal Methods — Find, Use, Conserve, Execute —

git: `https://gitlab.com/sosy-lab/benchmarking/fm-tools`
web: `https://fm-tools.sosy-lab.org`

Dirk Beyer
LMU Munich, Germany

May 28, 2025, at Fuzzing Summer School in Singapore

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

SoSy-Lab Software Systems

Vision

- All tools for formal methods work together to solve hard verification problems and make our world safer and more secure.
- Model checkers and theorem provers can be integrated into the software-development process as seamless as unit testing today.
- Model checkers, theorem provers, SMT solvers, and testers use common interfaces for interaction and composition.

Some Steps Towards the Vision

- **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)
- Limits

Maintained by formal-methods community

Example: Entry for CPAchecker

```yaml
id: cpachecker
name: CPAchecker
description: |
  CPAchecker is a configurable framework for software
     verification that
  is based on configurable program analysis and
  implements many model-checking algorithms
  to check for software errors and to verify program properties.
input_languages:
  - C
project_url: https://cpachecker.sosy-lab.org
repository_url: https://gitlab.com/sosy-lab/software/cpachecker
spdx_license_identifier: Apache-2.0
benchexec_toolinfo_module: benchexec.tools.cpachecker
fmtools_format_version: "2.0"
fmtools_entry_maintainers:
  - dbeyer
  - ricffb
  - PhilippWendler
```

Example: CPAchecker's Contacts

```yaml
maintainers:
  - orcid: 0000-0003-4832-7662
    name: Dirk Beyer
    institution: LMU Munich
    country: Germany
    url: https://www.sosy-lab.org/people/dbeyer/
  - orcid: 0000-0002-5139-341X
    name: Philipp Wendler
    institution: LMU Munich
    country: Germany
    url: https://www.sosy-lab.org/people/wendler/
```

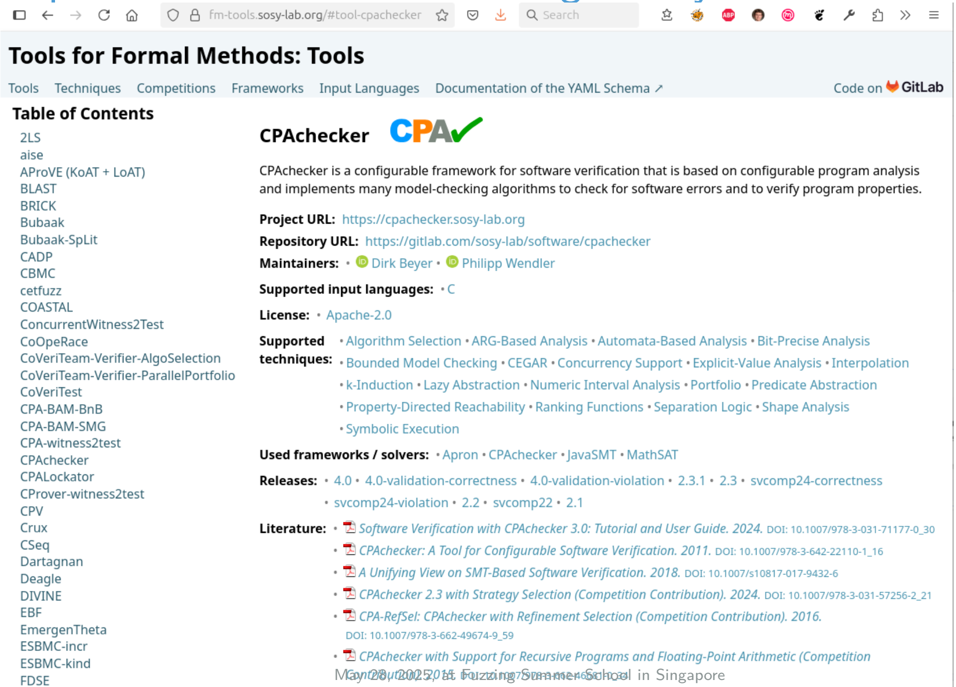
Example: CPAchecker's Versions

```
versions:
  - version: "4.0"
    doi: 10.5281/zenodo.14203369
    benchexec_toolinfo_options: ["--svcomp25", "--heap",
        "10000M", "--benchmark", "--timelimit", "900␣s"]
    required_ubuntu_packages:
      - openjdk-17-jdk-headless
    base_container_images:
      - docker.io/ubuntu:22.04
  - version: "4.0-validation-correctness"
    doi: 10.5281/zenodo.14203369
    benchexec_toolinfo_options: ["--witness", "${witness}",
        "--correctness-witness-validation", "--heap", "5000m",
        "--benchmark", "--option",
        "witness.checkProgramHash=false", "--option",
        "cpa.predicate.memoryAllocationsAlwaysSucceed=true"]
    required_ubuntu_packages:
      - openjdk-17-jdk-headless
    base_container_images:
      - docker.io/ubuntu:22.04
```

Example: CPAchecker's Documentation

```
literature:
  - doi: 10.1007/978-3-031-71177-0_30
    title: "Software␣Verification␣with␣CPAchecker␣3.0:␣Tutorial␣
        and␣User␣Guide”
    year: 2024
  - doi: 10.1007/978-3-642-22110-1_16
    title: "CPAchecker:␣A␣Tool␣for␣Configurable␣Software␣
        Verification”
    year: 2011
  - doi: 10.1007/s10817-017-9432-6
    title: "A␣Unifying␣View␣on␣SMT-Based␣Software␣Verification”
    year: 2018
```

Example: CPAchecker's Web-Page Entry



Tools for Formal Methods: Tools

Tools · Techniques · Competitions · Frameworks · Input Languages · Documentation of the YAML Schema ↗ · Code on GitLab

Table of Contents

CPAchecker

CPAchecker is a configurable framework for software verification that is based on configurable program analysis and implements many model-checking algorithms to check for software errors and to verify program properties.

Project URL: https://cpachecker.sosy-lab.org

Repository URL: https://gitlab.com/sosy-lab/software/cpachecker

Maintainers: • Dirk Beyer • Philipp Wendler

Supported input languages: • C

License: • Apache-2.0

Supported techniques: • Algorithm Selection • ARG-Based Analysis • Automata-Based Analysis • Bit-Precise Analysis • Bounded Model Checking • CEGAR • Concurrency Support • Explicit-Value Analysis • Interpolation • k-Induction • Lazy Abstraction • Numeric Interval Analysis • Portfolio • Predicate Abstraction • Property-Directed Reachability • Ranking Functions • Separation Logic • Shape Analysis • Symbolic Execution

Used frameworks / solvers: • Apron • CPAchecker • JavaSMT • MathSAT

Releases: • 4.0 • 4.0-validation-correctness • 4.0-validation-violation • 2.3.1 • 2.3 • svcomp24-correctness • svcomp24-violation • 2.2 • svcomp22 • 2.1

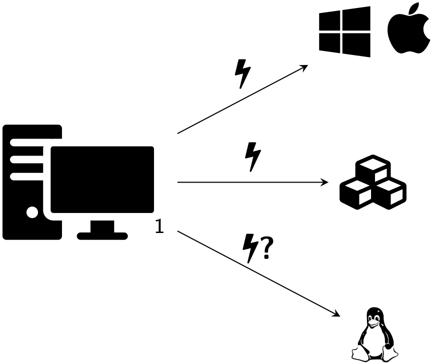
Literature:

- *Software Verification with CPAchecker 3.0: Tutorial and User Guide. 2024.* DOI: 10.1007/978-3-031-71177-0_30
- *CPAchecker: A Tool for Configurable Software Verification. 2011.* DOI: 10.1007/978-3-642-22110-1_16
- *A Unifying View on SMT-Based Software Verification. 2018.* DOI: 10.1007/s10817-017-9432-6
- *CPAchecker 2.3 with Strategy Selection (Competition Contribution). 2024.* DOI: 10.1007/978-3-031-57256-2_21
- *CPA-RefSel: CPAchecker with Refinement Selection (Competition Contribution). 2016.* DOI: 10.1007/978-3-662-49674-9_59
- *CPAchecker with Support for Recursive Programs and Floating-Point Arithmetic (Competition Contribution). 2015.* DOI: 10.1007/978-3-662-46681-0_34

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

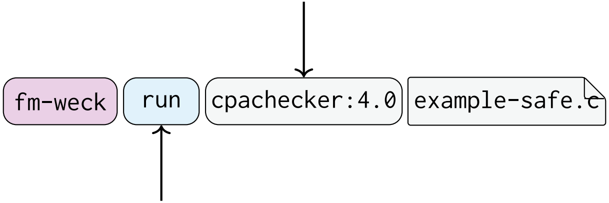
What about the Environment?



[1] Image: Flaticon.com

FM-WECK: Run Tools in Conserved Environment [2, Proc. FM 2024]

Refer to known fm-tools by name:version

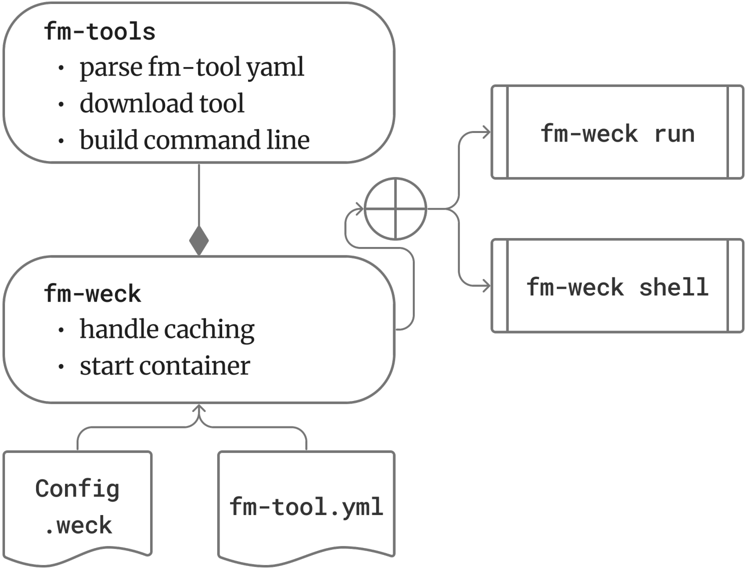


```
fm-weck run cpachecker:4.0 example-safe.c
```

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

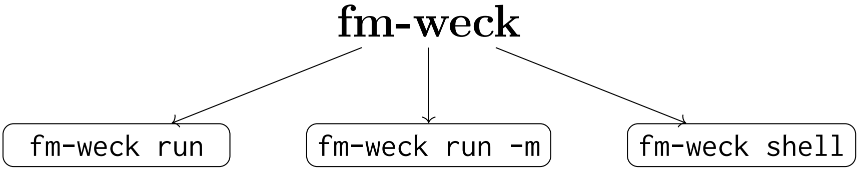


fm-tools

- parse fm-tool yaml
- download tool
- build command line

fm-weck

- handle caching
- start container



fm-weck

`fm-weck run`

`fm-weck run -m`

`fm-weck shell`

- 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

Conclusion FM-Tools and FM-Weck

FM-Tools collects and stores essential information to:

- Generate a knowledge base about formal-methods tools [1] `https://fm-tools.sosy-lab.org`
- Conserve tool versions and their required environment (with help by Zenodo and Podman/Docker)
- Run a tool in conserved environment via FM-Weck [2]
- Please add your tool

`https://fm-tools.sosy-lab.org`

Application: Competition on Software Testing

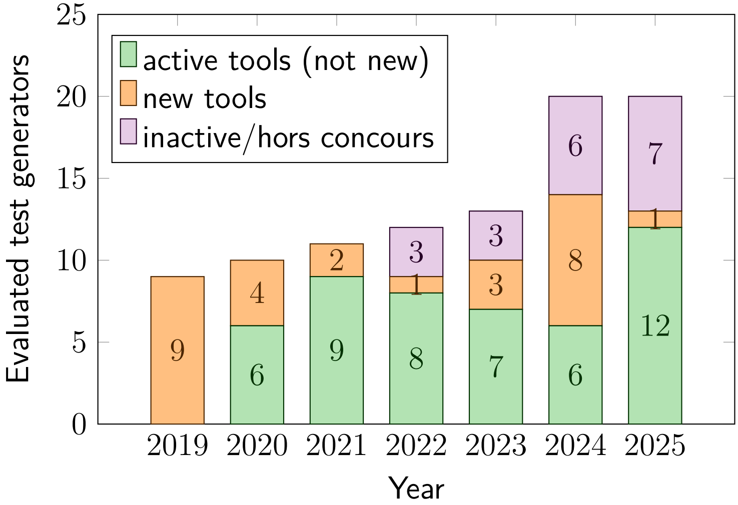
Report from 2025 [3, Proc. FASE]

Advances in Automatic Software Testing: Test-Comp 2025

Proc. FASE, Springer, 2025.
https://doi.org/10.1007/978-3-031-90900-9_13

Number of Participants

Number of evaluated test generators for each year (top: number of first-time participants; bottom: previous year's participants)



Motivation - Goals

1. Community suffers from unreproducible results
   → Establish set of benchmarks
2. Publicity for tools that are available
   → Provide state-of-the-art overview
3. Support the development of verification tools
   → Give credits and visibility to developers
4. Establish standards
   → Specification language, Test-suites,
   Benchmark definitions, Validators

Schedule of Sessions

Session Test-Comp:

- Competition Report, by organizer
- System Presentations, 10 min by each team
- Open Jury Meeting, Community Discussion

Procedure – Time Line

Three Steps – Three Deadlines:

- Benchmark submission deadline
- System submission
- Notification of results (approved by teams)

Test Problem

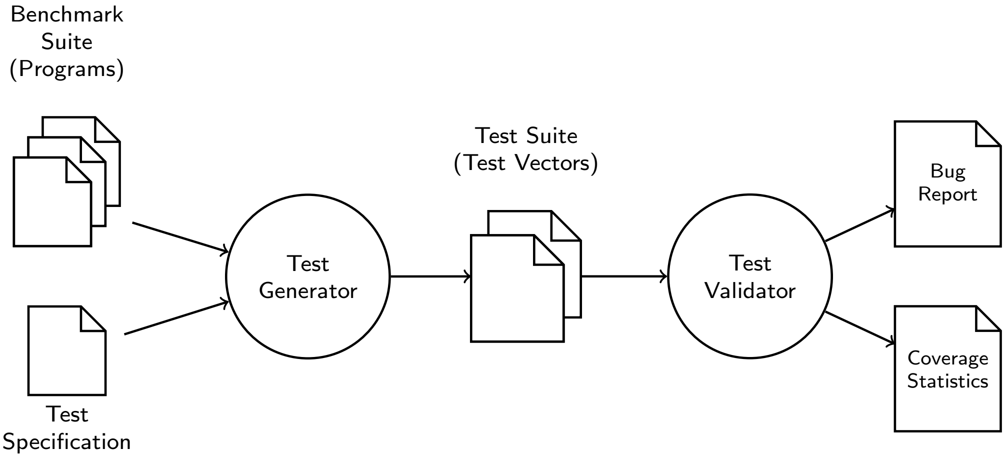
Input:

- C program → GNU/ANSI C standard
- Test Specification:
  → Coverage of function call
  → Branch coverage

Output:

- Test suite

Flow of the Test-Comp execution



Environment

Machines (1000 $ consumer machines):

- CPU: 3.4 GHz 64-bit Quad-Core CPU
- RAM: 33 GB
- OS: GNU/Linux (Ubuntu 24.04)

Resource limits:

- 15 GB memory
- 15 min CPU time
- 4 processing units

Scoring Schema (since 2019)

Common principles: Ranking measure should be

- easy to understand
- reproducible
- computable in isolation for one tool

Test-Comp:

- Coverage of call to function:
  1 point or 0 points
- Coverage of branches:
  TEST-COV coverage value (between 0 and 1)

Fair and Transparent

Jury:

- Team: one member of each participating candidate
- Term: one year (until next participants are determined)

Systems:

- All systems are available in open GitLab repo
- Configurations and Setup in GitLab repository
  → Integrity and reproducibility guaranteed

Competition Candidates

Qualification:

- 20 Qualified
- One person can participate with different tools
- One tool can participate with several configurations (frameworks, no tool-name inflation)

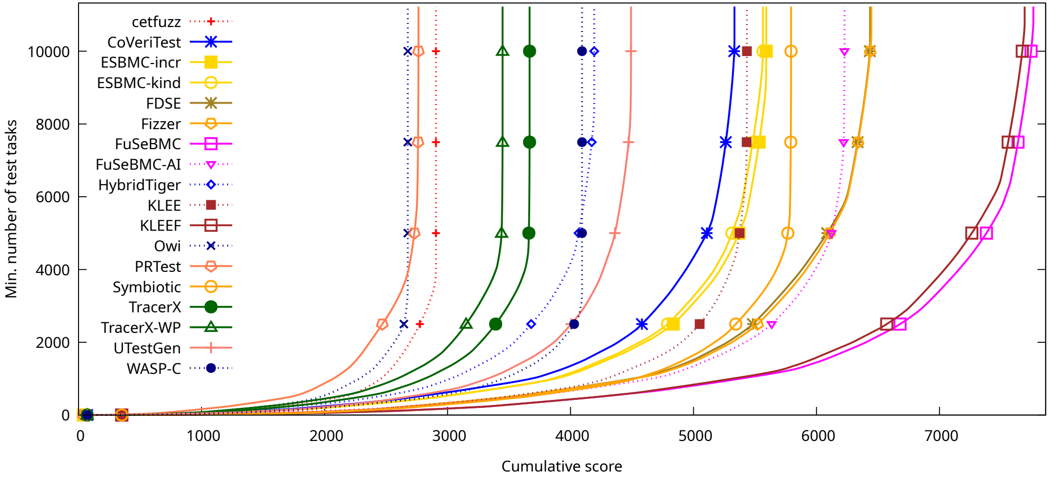
Benchmark quality:

- Community effort, documented on GitHub

Role of organizer:

- Just service: Advice, Technical Help, Executing Runs

Results – Example: Overall



Thanks to:

- Jury (12 people)
- 20 Tools evaluated
- FASE Steering Committee and PC Chairs
- Sponsors: LMU Munich and ETAPS

References I

[1] Beyer, D.: Find, use, and conserve tools for formal methods. In: Proc. Festschrift Podelski 65th Birthday. Springer (2024). https://www.sosy-lab.org/research/pub/2024-Podelski65.Find_Use_and_Conserve_Tools_for_Formal_Methods.pdf

[2] Beyer, D., Wachowitz, H.: FM-WECK: Containerized execution of formal-methods tools. In: Proc. FM. pp. 39–47. LNCS 14934, Springer (2024). doi:10.1007/978-3-031-71177-0_3

[3] Beyer, D.: Advances in automatic software testing: Test-Comp 2025. In: Proc. FASE. pp. 257–274. LNCS 15693, Springer (2025). doi:10.1007/978-3-031-90900-9_13