Find, Use, and Conserve Tools for Formal Methods

Dirk Beyer

LMU Munich, Munich, Germany

Abstract. The research area of formal methods has made enormous progress in the last 20 years, and many tools exist to apply formal methods to practical problems. Unfortunately, many of these tools are difficult to find and install, and often they are not executable due to missing installation requirements. The findability and wide adoption of tools, and the reproducibility of research results, could be improved if all major tools for formal methods were conserved and documented in a central repository of tools for formal methods (cf. FAIR principles).
This paper describes a solution to this problem: Collect and maintain essential data about tools for formal methods in a central repository, called FM-Tools, available at https://gitlab.com/sosy-lab/benchmarking/fm-tools. The repository contains metadata, such as which tools are available, which versions are advertized for each tool, and what command-line arguments to use for default usage. The actual tool executables are stored in tool archives at Zenodo, and for technically deep documentation, references point to archived publications or project web sites. Two communities, which are concerned with software verification and testing, already adopted the FM-Tools repository for their comparative evaluations.
Andreas Podelski and his research group, with their Ultimate family of tools for software verification, are among the early adopters of this strategy, and the Ultimate tools are included in the repository from its beginning.

Keywords: FAIR · Formal Methods · Long-Term Archiving · Reuse · Conservation · Reproducibility · Competitions · Software Tools · FM-Tools

1 Introduction

The research community of formal methods, especially formal verification of medium and large software systems, has seen a lot of progress in the past 20 years. As a result, there are many tool implementations available, and a recent survey [1], co-authored by Andreas Podelski, gives an overview of the milestones in the history of software model checking and the available tools for verification of C and Java programs. Those tools are mature in their performance and quality, as witnessed and regularly measured by comparative evaluations of the tools [2]. There are several competitions available in this research area of formal methods, for example, the competitions on satisfiability (SAT-COMP [3]), theorem proving (TPTP [4]), SMT solving (SMT-COMP [5]), software verification (SV-COMP [2], RERS [6] and VerifyThis [7]), software testing (Test-Comp [8]), and termination checking (termCOMP [9]). More competitions in the area of formal methods are explained in the TOOLympics 2019 report [10].

Besides all this great progress, and many success stories with formal verification of software in industry [11, 12, 13, 14, 15, 16, 17], users still have reasons to complain that the tools are difficult to find and install, and often they are not executable due to missing requirements [18, 19]. This hinders the wide application of these tools; often it is not possible to reproduce results reported in research publications. There is no standard way to find or conserve tools and components in this research area. Such tools, and metadata for the tools, should follow the FAIR principles (findable, accessible, interoperable, and reusable) [20, 21].

We propose a solution to this problem, by designing a data repository in which metadata about tools for formal methods are collected and maintained. The goal is to conserve a definition of how to execute each tool and what their requirements are. This central repository of tools for formal methods is hosted on GitLab at `https://gitlab.com/sosy-lab/benchmarking/fm-tools` and freely accessible, since all information is licensed under the Creative Commons license CC-BY 4.0.

The repository is already actively used to store metadata about the tools, their archive ids, their versions, and their documentation. Currently the two competitions on software verification (SV-COMP) and testing (Test-Comp) use the repository to track the participation in their comparative evaluations. For this use case, the repository specifies which tool version to execute for the competition and with which command-line options. Furthermore, the data contain information about whom to contact and who from the development team of the tool represents the team in the competition jury. The actual tool archives with the executables are stored at Zenodo and identified by DOIs. The data repository also provides pointers to documentation, archived in digital libraries or project web sites.

This topic fits well for this Festschrift, because Andreas Podelski, together with his team, participates since 2013 in the competition on software verification. The Ultimate family of tools for software verification and the development team participate in the community service around the competition and achieve top results, witnessed by the many medals that the team won. They are also concerned with making their tools available to others via an easy-to-use web interface. Often, such an excellent infrastructure can be offered only by large and strong development teams. This paper proposes to make *all* tools available via a web service.

The goal of this paper is to describe the above-mentioned data repository, outline how to make all tools available to users and machines via an easy-to-use web interface as well as command-line interface, and how to achieve a central store of (meta) information about tools for formal methods.

Contributions. This paper makes the following contributions:

- the repository FM-Tools as solution to the problem of collecting and maintaining data about tools for formal methods,
- a description of the repository's structure and its integration with the tools CoVeriTeam, CoVeriTeam-Service, and FM-Weck, and
- an artifact to try out executing a conserved tool.

Related Work. The web site Yahoda [22, 23] was created in 2002 to provide unified information about verification tools and to allow the developers of the tools to maintain the data about their tools. The last available version is from 2011 and contains 67 tool entries. The dataset ProVerB [24, 25] is also concerned with collecting information about tools for formal methods. The dataset contains a classification of tools and describes roughly the input and output of the tools, as well as some techniques that the tools support. The data set contains information about 384 tools, based on a systematic search, starting with the proceedings of the conferences CAV and TACAS. Maintaining the information is the main challenge in both cases: The web server for Yahoda is not reachable anymore, but thanks to the Internet Archive, the information is still available. The repository and web site of ProVerB was not touched since more than a year, but the data are also long-term archived at Zenodo. Our approach is to connect the maintenance of the data in the FM-Tools collection to regular comparative evaluations of the performance of tools (research competitions), and regularly publish FM-Tools snapshots in long-term archives at Zenodo [26].

This is not the first attempt to see formal-methods tools as components (cf. [27, 28, 29, 30]) or to provide them via a central web service [31, 32, 33, 34, 35]. Also, some tool projects provide specialized web services with their tools (e.g., Ultimate, CPAchecker). CoVeriTeam-Service [36] is more general: it provides a web service for almost all tools in the FM-Tools repository. The FM-Tools format (version 2) for describing tools can be seen as an extension of CoVeriTeam's format (version 1) for defining atomic actors [37, Listing 2]. With FM-Tools, we now add a lot more important information about the tools, in particular, it serves as the central location to announce the execution environment for a tool in form of container images in OCI image format and Ubuntu packages.

So far, no existing approach addresses the issue of conserving the tools and ensuring the tools' execution in the future.

2 The FM-Tools Repository

The current version of the FM-Tools repository `https://gitlab.com/sosy-lab/benchmarking/fm-tools` consists of the following directories (suffix '/') and files:

- presentations/ Contains presentations that describe the tools (see below).
- data/ Contains the main data about tools (see below).
- ci/ Contains scripts that ensure the consistency of the data. The scripts are executed by the continuous-integration (CI) pipelines of the GitLab repository.
- scripts/ Contains the contents of another repository as submodule. These scripts are used by the CI pipelines.
- CODEOWNERS Contains a specification of which file in the repository may be changed by which set of users. The file follows the GitLab format and is derived from the metadata for the tools.
- LICENSE.md Contains the license of all files in this repository.
- README.md Contains a description of the repository.

```yaml
name: UAutomizer
input_languages:
  - C
project_url: https://ultimate-pa.org
repository_url: https://github.com/ultimate-pa/ultimate
spdx_license_identifier: LGPL-3.0-or-later
benchexec_toolinfo_module: ultimateautomizer.py
fmtools_format_version: "2.0"
fmtools_entry_maintainers:
  - danieldietsch
```

Fig. 1: Data file for Ultimate Automizer — tool data

The directory **presentations/** contains presentations about tools described in **data/**. An entry consists of two files: `presentations/<tool-id>_<event-id>.pdf` and `presentations/<tool-id>_<event-id>.pdf.license`, where `<tool-id>` matches one of the file names `data/<tool-id>.yml` and `<event-id>` identifies the event where the presentation was given. The file with extension `.pdf` contains a presentation in PDF/A format; the file with extension `.license` contains an SPDX identifier of the license (for example, `SPDX-License-Identifier: CC-BY-4.0`).

In the following subsections, we focus on the directory **data/**. This directory contains the special file **schema.yml**, which defines the format of all other files in the directory: the tool descriptions. The names of the tool-description files are of the form `<tool-id>.yml`, where `tool-id` is an identifier that consists of lowercase letters, digits, and hyphens. In the following we define the contents of the tool-description files. For illustration, we use the example file **uautomizer.yml** for the tool Ultimate Automizer [38], one of Andreas Podelski's tools.

2.1 Tool Description

Figure 1 shows an example of the tool-data section of the file. The tool description starts with the key `name`, whose value is the (stylized) name of the tool. The YAML key `input_languages` has a list of languages as value, specifying which input formats the tool supports. The keys `project_url` and `repository_url` specify the project's web site and the source-code repository, respectively. The key `spdx_license_identifier` specifies the license of the tool in the standard SPDX format (`https://spdx.org/licenses/`). The key `benchexec_toolinfo_module` specifies the BenchExec [39] tool-info module that is necessary to assemble the command line for the tool's execution and to interpret the tool's output. The YAML key `fmtools_format_version` specifies the version of the tool-description format, currently `2.0`. The key `fmtools_entry_maintainers` specifies a list of maintainers of this tool description. The list elements must be valid GitLab user names, and those accounts will end up in the file `CODEOWNERS` in the top-level directory, in order to manage who can make changes to the data and approve merge requests.

```yaml
maintainers:
  - orcid: 0000-0003-4252-3558
    name: Matthias Heizmann
    institution: University of Freiburg
    country: Germany
    url: https://swt.informatik.uni-freiburg.de/staff/heizmann
  - orcid: 0000-0003-4885-0728
    name: Dominik Klumpp
    institution: University of Freiburg
    country: Germany
    url: https://swt.informatik.uni-freiburg.de/staff/klumpp
  - orcid: 0000-0002-5656-306X
    name: "Frank Schüssele"
    institution: University of Freiburg
    country: Germany
    url: https://swt.informatik.uni-freiburg.de/staff/schuessele
  - orcid: 0000-0002-8947-5373
    name: Daniel Dietsch
    institution: University of Freiburg
    country: Germany
    url: https://swt.informatik.uni-freiburg.de/staff/dietsch
```

Fig. 2: Data file for Ultimate Automizer — tool maintainers

2.2 Maintainers

Figure 2 shows an example of the tool maintainers. The key `maintainers` has a list of dictionaries as value. Each dictionary specifies one maintainer, by the keys `orcid`, `name`, `institution`, `country`, and homepage `url`.

2.3 Tool Versions

Figure 3 shows an example of the tool versions (we list only those from 2024 here). The key `versions` has a list of dictionaries as value. Each dictionary specifies one tool version, by the following keys: The key `version` is an identifier for a specific version of the tool, to be referred to, for example, for the definition which version participated in a competition. The key `doi` defines the tool archive. The DOI points to a specific version of the tool archives on Zenodo. The key `benchexec_toolinfo_options` specifies the command-line options that the developers define to be used to obtain optimal functionality. The key `required_ubuntu_packages` defines a list of Ubuntu packages that are required to be installed for the tool to work properly. The key `base_container_images` identifies a list of a container images in OCI image format (as used by Docker and Podman) in which the tool can be correctly executed *after installing* the packages defined under `required_ubuntu_packages`. The key `full_container_images` identifies a list of a container images in OCI image format in which the tool can be correctly executed *without* further installation of any packages. For example, Ultimate Automizer can be executed with the container image `registry.gitlab.com/sosy-lab/benchmarking/competition-scripts/user:2024` from the given domain and path without installing any package.

```yaml
versions:
  - version: svcomp24-correctness-post-deadline-yaml-wrapper-fix
    doi: 10.5281/zenodo.10223333
    benchexec_toolinfo_options:
      [--full-output, --witness-type, correctness_witness]
    required_ubuntu_packages:
      - openjdk-11-jre-headless
    base_container_images:
      - docker.io/ubuntu:22.04
    full_container_images:
      - registry.gitlab.com/sosy-lab/benchmarking/...-scripts/user:2024
  - version: svcomp24
    doi: 10.5281/zenodo.10203545
    benchexec_toolinfo_options: [--full-output]
    required_ubuntu_packages:
      - openjdk-11-jre-headless
    base_container_images:
      - docker.io/ubuntu:22.04
    full_container_images:
      - registry.gitlab.com/sosy-lab/benchmarking/...-scripts/user:2024
  - version: svcomp24-correctness
    doi: 10.5281/zenodo.10203545
    benchexec_toolinfo_options:
      [--full-output, --witness-type, correctness_witness]
    required_ubuntu_packages:
      - openjdk-11-jre-headless
    base_container_images:
      - docker.io/ubuntu:22.04
    full_container_images:
      - registry.gitlab.com/sosy-lab/benchmarking/...-scripts/user:2024
  - version: svcomp24-violation
    doi: 10.5281/zenodo.10203545
    benchexec_toolinfo_options:
      [--full-output, --witness-type, violation_witness]
    required_ubuntu_packages:
      - openjdk-11-jre-headless
    base_container_images:
      - docker.io/ubuntu:22.04
    full_container_images:
      - registry.gitlab.com/sosy-lab/benchmarking/...-scripts/user:2024
```

Fig. 3: Data file for Ultimate Automizer — tool versions (the identifier `registry.gitlab.com/sosy-lab/benchmarking/competition-scripts/user:2024` is abbreviated using ...)

2.4 Competition Participation

Figure 4 shows an example of the tool's participation declaration in the competition on software verification SV-COMP 2024 [2]. The corresponding YAML key `competition_participations` has a list of dictionaries as value. The key `competition` and `track` have as value the name of the competition and the name of the competition track for which this participation is meant, respectively. The key `tool_version` refers to a version of the tool that is defined under the key `versions` above. The key `jury_member` has a dictionary as value and defines the team member who represents this tool in the competition jury. The dictionary consists of entries with the keys `orcid`, `name`, `institution`, `country`, and `url`, where the URL refers to the person's home page. The declaration of the specific

```yaml
competition_participations:
  - competition: SV-COMP 2024
    track: Verification
    tool_version: svcomp24
    jury_member:
      orcid: 0000-0003-4252-3558
      name: Matthias Heizmann
      institution: University of Freiburg
      country: Germany
      url: https://swt.informatik.uni-freiburg.de/staff/heizmann
  - competition: SV-COMP 2024
    track: Validation of Correctness Witnesses 1.0
    tool_version: svcomp24-correctness
    jury_member:
      orcid: 0000-0003-4252-3558
      name: Matthias Heizmann
      institution: University of Freiburg
      country: Germany
      url: https://swt.informatik.uni-freiburg.de/staff/heizmann
  - competition: SV-COMP 2024
    track: Validation of Correctness Witnesses 2.0
    tool_version: svcomp24-correctness-post-deadline-yaml-wrapper-fix
    jury_member:
      orcid: 0000-0003-4252-3558
      name: Matthias Heizmann
      institution: University of Freiburg
      country: Germany
      url: https://swt.informatik.uni-freiburg.de/staff/heizmann
  - competition: SV-COMP 2024
    track: Validation of Violation Witnesses 1.0
    tool_version: svcomp24-violation
    jury_member:
      orcid: 0000-0003-4252-3558
      name: Matthias Heizmann
      institution: University of Freiburg
      country: Germany
      url: https://swt.informatik.uni-freiburg.de/staff/heizmann
```


Fig. 4: Data file for Ultimate Automizer — tool's competition participation

version, in particular via the DOI, together with the command-line options and the execution environment (container and packages) make it possible to execute this tool at any time, and reproduce the results obtained in the competition.

2.5 Documentation

Figure 5 shows an example of the documentation of the tool. The key `techniques` has a list of keywords as value, where each keyword refers to an established technique in software model checking. The key `literature` has a list of dictionaries as value. Each literature dictionary has the keys `doi` to identify the literature document (documents without DOI can be mentioned on the project web site), `title` to mention the title of the document, and `year` to mention the year of publication (the last two values are implied by the DOI, but are mentioned here to have the data human readable, consistency should be ensure via CI).

```yaml
techniques:
  - CEGAR
  - Predicate Abstraction
  - Bit-Precise Analysis
  - Lazy Abstraction
  - Interpolation
  - Automata-Based Analysis
  - Concurrency Support
  - Ranking Functions
  - Algorithm Selection
  - Portfolio

literature:
  - doi: 10.1007/978-3-642-39799-8_2
    title: "Software Model Checking for People Who Love Automata"
    year: 2013
  - doi: 10.1007/978-3-031-30820-8_39
    title: "Ultimate Automizer 2023 (Competition Contribution)"
    year: 2023
```

Fig. 5: Data file for Ultimate Automizer — tool documentation

2.6 FAIR Principles of the FM-Tools Repository

At the time of writing, FM-Tools describes and captures metadata of 91 tools from the formal-methods research community. The repository is hosted at GitLab. FM-Tools is an open-source data repository that follows the FAIR principles [20, 21] (findable, accessible, interoperable, reusable).

F: The repository FM-Tools is searchable on the public internet, and mirrored by the software archive Software Heritage. A human readable web site with the most important information is continuously generated from the repository.

A: All files of FM-Tools are retrievable using the HTTP protocol via the GitLab web service. Authentication is not necessary for reading the data (change requests require a GitLab account). If GitLab should become unavailable, the snapshots at Zenodo and the mirror at Software Heritage are still available.

I: The format of the tool entries in FM-Tools is defined using a YAML schema, and continuous-integration pipelines check the syntax and context conditions. The tool archives are identified using DOIs (pointing to the archives' landing page at Zenodo) and the researchers are identified using ORCIDs (pointing to the researchers' profiles at ORCID).

R: The license of the FM-Tools data is CC-BY 4.0. The data represented in FM-Tools meet the community standards of the competitions SV-COMP (`https://sv-comp.sosy-lab.org`) and Test-Comp (`https://test-comp.sosy-lab.org`). It is under the researchers' control how much they reveal about themselves on ORCID, and under which license the tool is available on Zenodo.

3 Integration of the FM-Tools Repository with CoVeriTeam and FM-Weck

The data in the FM-Tools repository can be used to conveniently execute tools for formal methods, without the need to install them or to take measures such as containers to ensure isolated execution without security risks on the user machine.

3.1 Tool Execution via CoVeriTeam Service

CoVeriTeam [37] is a tool that, among other things, automates the download, installation, and execution of tools in safe and secure environments based on BenchExec [39]. CoVeriTeam-Service [36] is a service that offers the features of CoVeriTeam as a web service, that is, the input files are sent to a remote server, the tool is executed on the remote server, and results are fetched from the remote server and delivered back to the user. Since CoVeriTeam Service executes formal-methods tools remotely without locally installing them, this way of execution is ideal for continuous integration, because the actual automated-reasoning work is offloaded to a remote compute server.

In the following we assume that the repository for CoVeriTeam is cloned using

```
git clone --recurse-submodules \
  git@gitlab.com:sosy-lab/software/coveriteam.git
```

and that the current directory is the main directory of that checkout. A full command line to execute Ultimate Automizer remotely, using the specific version `svcomp24` of the tool, the specification `no-overflow`, and the particular program `AdditionIntMax.i` would look as follows:

```
bin/verify --remote -t uautomizer -v svcomp24 --spec no-overflow \
  examples/test-data/c/AdditionIntMax.i
```

This command is executed on the remote server (by default, CoVeriTeam Service uses the server `coveriteam-service.sosy-lab.org`), which downloads, installs, and executes the tool Ultimate Automizer in version `svcomp24`, and returns the result to the user. The verdict from Ultimate Automizer is reported as `false(no-overflow)`, that is, an overflow of a variable of type signed integer happens. More detailed information can be found in the CoVeriTeam output folder `cvt-output/`, in particular, the execution trace with measurements of consumed resources (such as CPU time, wall time, and memory), the complete log of the tool's output to stdout, and possibly a verification witness.

3.2 Tool Execution via FM-Weck

FM-Weck [40] provides support for conveniently running tools from the FM-Tools repository in their designated OCI containers: it downloads the tool and its container, and executes the tool in its container (in isolation). The tool has two modes: (i) FM-Weck takes a base container image specified as a value in the list for key `base_container_images` of the version dictionary in the `versions` section, installs the packages listed in the key `required_ubuntu_-packages`, and executes the tool available in the archive specified by the value for key `doi`. (ii) FM-Weck takes a full container image specified as a value in the list for key `full_container_images` of the version dictionary in the `versions` section, and executes the tool available in the archive specified by the value for key `doi`, without installing any packages.

FM-Weck enables all users to take advantage of the conserved tools as defined in the FM-Tools repository. This approach makes it possible to execute the formal-methods tools hopefully after many years, when a machine with the required operating system and packages cannot be found anymore. The container images will still be available, and thus, the tools still be executable. Currently, it is an open challenge for the formal-methods community to achieve reproducibility of any experimental results obtained with tools and published articles. Full reproducibility is not yet achieved up to now, partially because tools are not easy to execute, which is the concern that FM-Weck tries to address.

We now explain how to execute UAutomizer via FM-Weck. We assume in the following a GNU/Linux machine with a working Podman installation. First, we install FM-Weck via pip:

```
pip install fm-weck
```

Assuming that `AdditionIntMax.i` and `no-overflow.prp` are in the current directory, the following command line downloads and executes UAutomizer in version `svcomp24` in a container specified under `full_container_images` in the FM-Tools record for version `svcomp24`:

```
fm-weck run uautomizer:svcomp24 \
  --property no-overflow.prp --data-model LP64 AdditionIntMax.i
```

The output of UAutomizer is the same as above. The command-line arguments and input files to UAutomizer can be adjusted using the option `-m` for FM-Weck.

4 Overview Web Site Generated from the Data

The community suffers from the situation that there is no central point of information that brings together all information necessary to understand which tools for what purpose are available. Thanks to the FM-Tools repository, we are able to generate a web site that lists the tools, and for each tool an information section displaying all interesting data, such as contact developers, documentation, supported techniques, where to find the tool archives, and which competitions used the tool to provide comparative results on effectivity and efficiency of the tool. The web site generated from the FM-Tools repository is available at https://fm-tools.sosy-lab.org.

For example, if we are interested in learning which tools use the technique CEGAR, then we can visit the above web site, select 'Techniques' from the menu and 'CEGAR' from the table of contents, to receive an overview of this technique and the tools supporting it. The overview starts with a short description of the technique CEGAR,

> "CounterExample-Guided Abstraction Refinement is a model-checking technique that iteratively refines the abstract model of the transition system by analyzing spurious counterexamples and learning a more precise abstraction."

followed by a list of literature references [41, 42, 43] and a list of tools using the technique CEGAR: BRICK [44], CoVeriTeam-Verifier-AlgoSelection [37, 45], CoVeriTeam-Verifier-ParallelPortfolio [37, 45], CoVeriTest [46, 47], CPA-BAM-BnB [48, 49], CPAchecker [50, 51], CPALockator [52, 53], Gazer-Theta [54, 55], Graves-CPA [56], HybridTiger [57, 58], JayHorn [59, 60], PeSCo-CPA [61, 62], PIChecker [63], Theta [64, 65], UAutomizer [38, 66], UGemCutter [67, 68], UKojak [69, 70], UTaipan [71, 72], UTestGen [73], VeriAbs [74, 75], and VeriAbsL [76].

5 Conclusion

This article defines a standard format for the collection and maintenance of important information about tools for formal methods. The goal is to address challenges with regards to findability, reusability, reproducibility, and conservation. (i) Findability is addressed by having a standard format that is used by a significant amount of users. Currently the repository contains over 90 tools for formal methods, mainly from the research communities in the area of software verification and testing, around the competitions SV-COMP and Test-Comp. (ii) Reusability is addressed by having contact information, data about various versions, and documentation available. (iii) Reproducibility is addressed by having authoritative archives with tool executables identified via DOIs, providing developer-recommended command-line parameters to be used with the tool, and the tool archives are long-term published at Zenodo. (iv) Conservation is addressed by capturing the execution environment, which consists of a defined list of container images in OCI image format and a list of required packages.

We hope that the ideas in this paper help towards addressing the grand challenge of reproducibility of experimental results that are obtained with formal-methods tools and published in the formal-methods proceedings and journals. We congratulate Andreas Podelski to his 65th birthday and especially to the many tools that were developed under his guidance — we are happy that we can contribute to conserving such excellent research products for future generations.

Data-Availability Statement. The FM-Tools repository is available open access (CC-BY 4.0) at `https://gitlab.com/sosy-lab/benchmarking/fm-tools`. Important versions of the FM-Tools repository are archived at Zenodo [26] (current version is 2.0). The repository is also mirrored at Software Heritage. The generated web site `https://fm-tools.sosy-lab.org/` is also mirrored at the Internet Archive. The artifact supporting this paper is available at Zenodo [77] and contains a snapshot of the CoVeriTeam repository in version 1.2.3 (such that CoVeriTeam Service can be executed) and a snapshot of the FM-Tools repository in version 2.0 as submodule (such that all data can be inspected).

Funding Statement. FM-Weck was supported by Deutsche Forschungsgemeinschaft (DFG) – 378803395 (ConVeY). CoVeriTeam and CoVeriTeam Service were supported by Deutsche Forschungsgemeinschaft (DFG) – 418257054 (Coop) and 378803395 (ConVeY).

Acknowledgements. We thank Sudeep Kanav and Henrik Wachowitz for the joint work on CoVeriTeam [37], CoVeriTeam Service [36], and FM-Weck [40].

References

1. Beyer, D., Podelski, A.: Software model checking: 20 years and beyond. In: Principles of Systems Design. pp. 554–582. LNCS 13660, Springer (2022). https://doi.org/10.1007/978-3-031-22337-2_27
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. Järvisalo, M., Berre, D.L., Roussel, O., Simon, L.: The international SAT solver competitions. AI Magazine **33**(1) (2012). https://doi.org/10.1609/aimag.v33i1.2395
4. Sutcliffe, G.: The CADE ATP system competition: CASC. AI Magazine **37**(2), 99–101 (2016). https://doi.org/10.1609/aimag.v37i2.2620
5. Weber, T., Conchon, S., Déharbe, D., Heizmann, M., Niemetz, A., Reger, G.: The SMT competition 2015-2018. J. Satisf. Boolean Model. Comput. **11**(1), 221–259 (2019). https://doi.org/10.3233/SAT190123
6. Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D., Păsăreanu, C.S.: Rigorous examination of reactive systems. The RERS challenges 2012 and 2013. Int. J. Softw. Tools Technol. Transfer **16**(5), 457–464 (2014). https://doi.org/10.1007/s10009-014-0337-y
7. Ernst, G., Huisman, M., Mostowski, W., Ulbrich, M.: VerifyThis: Verification competition with a human factor. In: Proc. TACAS. pp. 176–195. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_12
8. Beyer, D.: Software testing: 5th comparative evaluation: Test-Comp 2023. In: Proc. FASE. pp. 309–323. LNCS 13991, Springer (2023). https://doi.org/10.1007/978-3-031-30826-0_17
9. Giesl, J., Mesnard, F., Rubio, A., Thiemann, R., Waldmann, J.: Termination competition (termCOMP 2015). In: Proc. CADE. pp. 105–108. LNCS 9195, Springer (2015). https://doi.org/10.1007/978-3-319-21401-6_6
10. Bartocci, E., Beyer, D., Black, P.E., Fedyukovich, G., Garavel, H., Hartmanns, A., Huisman, M., Kordon, F., Nagele, J., Sighireanu, M., Steffen, B., Suda, M., Sutcliffe, G., Weber, T., Yamada, A.: TOOLympics 2019: An overview of competitions in formal methods. In: Proc. TACAS (3). pp. 3–24. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_1
11. Ball, T., Levin, V., Rajamani, S.K.: A decade of software model checking with Slam. Commun. ACM **54**(7), 68–76 (2011). https://doi.org/10.1145/1965724.1965743
12. Ball, T., Cook, B., Levin, V., Rajamani, S.K.: Slam and Static Driver Verifier: Technology transfer of formal methods inside Microsoft. In: Proc. IFM. pp. 1–20. LNCS 2999, Springer (2004). https://doi.org/10.1007/978-3-540-24756-2_1

13. Khoroshilov, A.V., Mutilin, V.S., Petrenko, A.K., Zakharov, V.: Establishing Linux driver verification process. In: Proc. Ershov Memorial Conference. pp. 165–176. LNCS 5947, Springer (2009). https://doi.org/10.1007/978-3-642-11486-1_14
14. Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., O'Hearn, P., 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
15. Cook, B.: Formal reasoning about the security of Amazon web services. In: Proc. CAV (2). pp. 38–47. LNCS 10981, Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_3
16. Chong, N., Cook, B., Kallas, K., Khazem, K., Monteiro, F.R., Schwartz-Narbonne, D., Tasiran, S., Tautschnig, M., Tuttle, M.R.: Code-level model checking in the software development workflow. In: Proc. ICSE. pp. 11–20. ICSE-SEIP '20, ACM (2020). https://doi.org/10.1145/3377813.3381347
17. Darke, P., Metta, R., Medicherla, R.K., Venkatesh, R.: Impactful research and tooling for program correctness. Commun. ACM **65**(11), 52–53 (October 2022). https://doi.org/10.1145/3551665
18. Alglave, J., Donaldson, A.F., Kröning, D., Tautschnig, M.: Making software verification tools really work. In: Proc. ATVA. pp. 28–42. LNCS 6996, Springer (2011). https://doi.org/10.1007/978-3-642-24372-1_3
19. Garavel, H., ter Beek, M.H., van de Pol, J.: The 2020 expert survey on formal methods. In: Proc. FMICS. pp. 3–69. LNCS 12327, Springer (2020). https://doi.org/10.1007/978-3-030-58298-2_1
20. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.W., d. Silva Santos, L.B., Bourne, P.E., Bouwman, J., Brookes, A.J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C.T., Finkers, R., Gonzalez-Beltran, A., Gray, A.J.G., Groth, P., Goble, C., Grethe, J.S., Heringa, J., 't Hoen, P.A.C., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S.J., Martone, M.E., Mons, A., Packer, A.L., Persson, B., Rocca-Serra, P., Roos, M., v. Schaik, R., Sansone, S.A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M.A., Thompson, M., v. d. Lei, J., v. Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J., Mons, B.: The FAIR guiding principles for scientific data management and stewardship. Sci. Data **3** (2016). https://doi.org/10.1038/sdata.2016.18
21. Jacobsen, A., d. Miranda Azevedo, R., Juty, N.S., Batista, D., Coles, S.J., Cornet, R., Courtot, M., Crosas, M., Dumontier, M., Evelo, C.T.A., Goble, C.A., Guizzardi, G., Hansen, K.K., Hasnain, A., Hettne, K.M., Heringa, J., Hooft, R.W.W., Imming, M., Jeffery, K.G., Kaliyaperumal, R., Kersloot, M.G., Kirkpatrick, C.R., Kuhn, T., Labastida, I., Magagna, B., McQuilton, P., Meyers, N., Montesanti, A., v. Reisen, M., Rocca-Serra, P., Pergl, R., Sansone, S.A., d. Silva Santos, L.O.B., Schneider, J., Strawn, G.O., Thompson, M., Waagmeester, A., Weigel, T., Wilkinson, M.D., Willighagen, E.L., Wittenburg, P., Roos, M., Mons, B., Schultes, E.: FAIR principles: Interpretations and implementation considerations. Data Intell. **2**(1-2), 10–29 (2020). https://doi.org/10.1162/DINT_R_00024
22. Crhová, J., Krčál, P., Strejček, J., Šafránek, D., Šimeček, P.: Yahoda: Verification tools database. In: Proc. Tools Day. pp. 99–103. FI MU Report Series FIMU-RS-2002-05, Masaryk University (2002)
23. Crhová, J., Krcál, P., Strejček, J., Šafránek, D., Simecek, P.: Yahoda: Verification tools database. `http://www.fi.muni.cz/yahoda/` (2002), [Available in the Inter-

net Archive at https://web.archive.org/web/20111119200847/http://anna.fi.muni.cz/yahoda]
24. Lathouwers, S., Zaytsev, V.: Modelling program-verification tools for software engineers. In: Proc. MODELS. pp. 98–108. ACM (2022). https://doi.org/10.1145/3550355.3552426
25. Lathouwers, S., Zaytsev, V.: Proverb: Dataset of tools and formats for program verification. Zenodo (2024). https://doi.org/10.5281/zenodo.10806218
26. Beyer, D.: FM-Tools releases. Zenodo. https://doi.org/10.5281/zenodo.10669734
27. Spector, A.Z.: Invited talk: Modular architectures for distributed and databases systems. In: Proc. PODS. pp. 217–224. ACM (1989). https://doi.org/10.1145/73721.73743
28. Giunchiglia, E., Narizzano, M., Tacchella, A., Vardi, M.Y.: Towards an efficient library for SAT: A manifesto. Electronic Notes in Discrete Mathematics **9**, 290–310 (2001). https://doi.org/10.1016/S1571-0653(04)00329-4
29. Shankar, N.: Little engines of proof. In: Proc. FME. pp. 1–20. LNCS 2391, Springer (2002). https://doi.org/10.1007/3-540-45614-7_1
30. Lampson, B.: Software components: Only the giants survive. In: Computer Systems. Monographs in Computer Science. pp. 137–145. Springer (2004). https://doi.org/10.1007/0-387-21821-1_21
31. Steffen, B., Margaria, T., Braun, V.: The Electronic Tool Integration platform: Concepts and design. STTT **1**(1-2), 9–30 (1997). https://doi.org/10.1007/s100090050003
32. Margaria, T., Nagel, R., Steffen, B.: jETI: A tool for remote tool integration. In: Proc. TACAS. pp. 557–562. LNCS 3440, Springer (2005). https://doi.org/10.1007/978-3-540-31980-1_38
33. Margaria, T., Nagel, R., Steffen, B.: Remote integration and coordination of verification tools in jETI. In: Proc. ECBS. pp. 431–436 (2005). https://doi.org/10.1109/ECBS.2005.59
34. Margaria, T.: Web services-based tool-integration in the ETI platform. Software and Systems Modeling **4**(2), 141–156 (2005). https://doi.org/10.1007/s10270-004-0072-z
35. Steffen, B.: The physics of software tools: SWOT analysis and vision. Int. J. Softw. Tools Technol. Transf. **19**(1), 1–7 (2017). https://doi.org/10.1007/s10009-016-0446-x
36. 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
37. Beyer, D., Kanav, S.: CoVeriTeam: On-demand composition of cooperative verification systems. In: Proc. TACAS. pp. 561–579. LNCS 13243, Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_31
38. Heizmann, M., Hoenicke, J., Podelski, A.: Software model checking for people who love automata. In: Proc. CAV. pp. 36–52. LNCS 8044, Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_2
39. Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: Requirements and solutions. Int. J. Softw. Tools Technol. Transfer **21**(1), 1–29 (2019). https://doi.org/10.1007/s10009-017-0469-y
40. Beyer, D., Wachowitz, H.: FM-Weck: Containerized execution of formal-methods tools. In: Proc. FM. LNCS, Springer (2024)
41. 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
42. Ball, T., Rajamani, S.K.: Boolean programs: A model and process for software analysis. Tech. Rep. MSR Tech. Rep. 2000-14, Microsoft Research (2000), https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2000-14.pdf

43. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003). https://doi.org/10.1145/876638.876643
44. Bu, L., Xie, Z., Lyu, L., Li, Y., Guo, X., Zhao, J., Li, X.: Brick: Path enumeration-based bounded reachability checking of C programs (competition contribution). In: Proc. TACAS (2). pp. 408–412. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_22
45. Beyer, D., Kanav, S., Richter, C.: Construction of verifier combinations based on off-the-shelf verifiers. In: Proc. FASE. pp. 49–70. Springer (2022). https://doi.org/10.1007/978-3-030-99429-7_3
46. Jakobs, M.C., Richter, C.: CoVeriTest with adaptive time scheduling (competition contribution). In: Proc. FASE. pp. 358–362. LNCS 12649, Springer (2021). https://doi.org/10.1007/978-3-030-71500-7_18
47. Beyer, D., Jakobs, M.C.: CoVeriTest: Cooperative verifier-based testing. In: Proc. FASE. pp. 389–408. LNCS 11424, Springer (2019). https://doi.org/10.1007/978-3-030-16722-6_23
48. Andrianov, P., Friedberger, K., Mandrykin, M.U., Mutilin, V.S., Volkov, A.: CPA-BAM-BnB: Block-abstraction memoization and region-based memory models for predicate abstractions (competition contribution). In: Proc. TACAS. pp. 355–359. LNCS 10206, Springer (2017). https://doi.org/10.1007/978-3-662-54580-5_22
49. Volkov, A.R., Mandrykin, M.U.: Predicate abstractions memory modeling method with separation into disjoint regions. Proceedings of the Institute for System Programming (ISPRAS) **29**, 203–216 (2017). https://doi.org/10.15514/ISPRAS-2017-29(4)-13
50. Baier, D., Beyer, D., Chien, P.C., Jankola, M., Kettl, M., Lee, N.Z., Lemberger, T., Lingsch-Rosenfeld, M., Spiessl, M., Wachowitz, H., Wendler, P.: CPAchecker 2.3 with strategy selection (competition contribution). In: Proc. TACAS (3). pp. 359–364. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_21
51. 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
52. Andrianov, P., Mutilin, V., Khoroshilov, A.: CPALockator: Thread-modular approach with projections (competition contribution). In: Proc. TACAS (2). pp. 423–427. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_25
53. Andrianov, P.S.: Analysis of correct synchronization of operating system components. Program. Comput. Softw. **46**, 712–730 (2020). https://doi.org/10.1134/S0361768820080022
54. Ádám, Zs., Sallai, Gy., Hajdu, Á.: Gazer-Theta: LLVM-based verifier portfolio with BMC/CEGAR (competition contribution). In: Proc. TACAS (2). pp. 433–437. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_27
55. Hajdu, Á., Micskei, Z.: Efficient strategies for CEGAR-based model checking. J. Autom. Reasoning **64**(6), 1051–1091 (2020). https://doi.org/10.1007/s10817-019-09535-x
56. Leeson, W., Dwyer, M.: Graves-CPA: A graph-attention verifier selector (competition contribution). In: Proc. TACAS (2). pp. 440–445. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_28
57. Ruland, S., Lochau, M., Jakobs, M.C.: HybridTiger: Hybrid model checking and domination-based partitioning for efficient multi-goal test-suite generation (competition contribution). In: Proc. FASE. pp. 520–524. LNCS 12076, Springer (2020). https://doi.org/10.1007/978-3-030-45234-6_26

58. Bürdek, J., Lochau, M., Bauregger, S., Holzer, A., von Rhein, A., Apel, S., Beyer, D.: Facilitating reuse in multi-goal test-suite generation for software product lines. In: Proc. FASE. pp. 84–99. LNCS 9033, Springer (2015). https://doi.org/10.1007/978-3-662-46675-9_6
59. Shamakhi, A., Hojjat, H., Rümmer, P.: Towards string support in JayHorn (competition contribution). In: Proc. TACAS (2). pp. 443–447. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_29
60. Kahsai, T., Rümmer, P., Sanchez, H., Schäf, M.: JayHorn: A framework for verifying Java programs. In: Proc. CAV. pp. 352–358. LNCS 9779, Springer (2016). https://doi.org/10.1007/978-3-319-41528-4_19
61. Richter, C., Wehrheim, H.: PeSCo: Predicting sequential combinations of verifiers (competition contribution). In: Proc. TACAS (3). pp. 229–233. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_19
62. Richter, C., Hüllermeier, E., Jakobs, M.C., Wehrheim, H.: Algorithm selection for software validation based on graph kernels. Autom. Softw. Eng. **27**(1), 153–186 (2020). https://doi.org/10.1007/s10515-020-00270-x
63. Su, J., Yang, Z., Xing, H., Yang, J., Tian, C., Duan, Z.: PIChecker: A POR and interpolation-based verifier for concurrent programs (competition contribution). In: Proc. TACAS (2). pp. 571–576. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_38
64. Bajczi, L., Telbisz, C., Somorjai, M., Ádám, Z., Dobos-Kovács, M., Szekeres, D., Mondok, M., Molnár, V.: Theta: Abstraction based techniques for verifying concurrency (competition contribution). In: Proc. TACAS (3). pp. 412–417. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_30
65. Tóth, T., Hajdu, A., Vörös, A., Micskei, Z., Majzik, I.: Theta: A framework for abstraction refinement-based model checking. In: Proc. FMCAD. pp. 176–179 (2017). https://doi.org/10.23919/FMCAD.2017.8102257
66. Heizmann, M., Bentele, M., Dietsch, D., Jiang, X., Klumpp, D., Schüssele, F., Podelski, A.: Ultimate automizer and the abstraction of bitwise operations (competition contribution). In: Proc. TACAS (3). pp. 418–423. LNCS 14572, Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_31
67. Klumpp, D., Dietsch, D., Heizmann, M., Schüssele, F., Ebbinghaus, M., Farzan, A., Podelski, A.: Ultimate GemCutter and the axes of generalization (competition contribution). In: Proc. TACAS (2). pp. 479–483. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_35
68. Farzan, A., Klumpp, D., Podelski, A.: Sound sequentialization for concurrent program verification. In: Proc. PLDI. pp. 506–521. ACM (2022). https://doi.org/10.1145/3519939.3523727
69. Nutz, A., Dietsch, D., Mohamed, M.M., Podelski, A.: Ultimate Kojak with memory safety checks (competition contribution). In: Proc. TACAS. pp. 458–460. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_44
70. Ermis, E., Hoenicke, J., Podelski, A.: Splitting via interpolants. In: Proc. VMCAI. pp. 186–201. LNCS 7148, Springer (2012). https://doi.org/10.1007/978-3-642-27940-9_13
71. Dietsch, D., Heizmann, M., Klumpp, D., Schüssele, F., Podelski, A.: Ultimate Taipan 2023 (competition contribution). In: Proc. TACAS (2). pp. 582–587. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_40
72. Greitschus, M., Dietsch, D., Podelski, A.: Loop invariants from counterexamples. In: Proc. SAS. pp. 128–147. LNCS 10422, Springer (2017). https://doi.org/10.1007/978-3-319-66706-5_7

73. Barth, M., Dietsch, D., Heizmann, M., Jakobs, M.C.: Ultimate TestGen: Test case generation with automata-based software model checking (competition contribution). In: Proc. FASE. pp. 326–330. LNCS 14573, Springer (2024). https://doi.org/10.1007/978-3-031-57259-3_20
74. Darke, P., Agrawal, S., Venkatesh, R.: VeriAbs: A tool for scalable verification by abstraction (competition contribution). In: Proc. TACAS (2). pp. 458–462. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_32
75. Afzal, M., Asia, A., Chauhan, A., Chimdyalwar, B., Darke, P., Datar, A., Kumar, S., Venkatesh, R.: VeriAbs: Verification by abstraction and test generation. In: Proc. ASE. pp. 1138–1141. IEEE (2019). https://doi.org/10.1109/ASE.2019.00121
76. Darke, P., Chimdyalwar, B., Agrawal, S., Venkatesh, R., Chakraborty, S., Kumar, S.: VeriAbsL: Scalable verification by abstraction and strategy prediction (competition contribution). In: Proc. TACAS (2). pp. 588–593. LNCS 13994, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_41
77. Beyer, D., Wachowitz, H.: CoVeriTeam release 1.2.3 (with FM-Tools 2.0). Zenodo (2024). https://doi.org/10.5281/zenodo.11193820